

AFIT/GE/ENG/99M-01

AN IMPROVED ASYNCHRONOUS IMPLEMENTATION
OF A FAST FOURIER TRANSFORM ARCHITECTURE
FOR SPACE APPLICATIONS

THESIS

David James Barnhart
Captain, USAF

AFIT/GE/ENG/99M-01

1 19990413 082

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the United States Government.

AN IMPROVED ASYNCHRONOUS IMPLEMENTATION
OF A FAST FOURIER TRANSFORM ARCHITECTURE
FOR SPACE APPLICATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

David James Barnhart, B.S.E.E.
Captain, USAF

March 1999

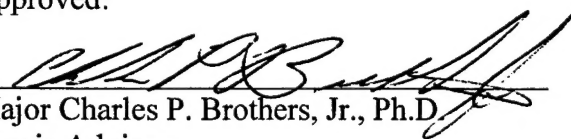
Approved for public release; distribution unlimited

AN IMPROVED ASYNCHRONOUS IMPLEMENTATION
OF A FAST FOURIER TRANSFORM ARCHITECTURE
FOR SPACE APPLICATIONS

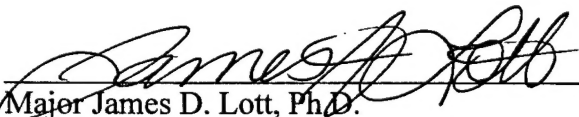
David James Barnhart, B.S.E.E.

Captain, USAF

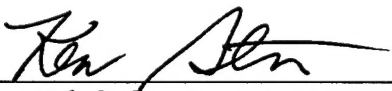
Approved:


Major Charles P. Brothers, Jr., Ph.D.
Thesis Advisor

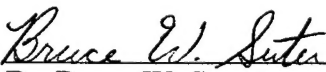
09 Mar 99
Date


Major James D. Lott, Ph.D.
Committee Member

09 MARCH 1999
Date


Dr. Kenneth S. Stevens
Committee Member

6-Mar-99
Date


Dr. Bruce W. Suter
Committee Member

25 February 1999
Date

Acknowledgments

Considering that a master's thesis cannot be accomplished without help from others, I have many people that I would like to thank for their support, guidance and encouragement during my assignment at AFIT.

I would first like to give Jesus Christ the glory and credit for the good things that He has done through me. May all know His saving grace! (John 3:16). I must also thank my loving wife Vonda for sacrificing so much for me to achieve so little. Her faithful support of me, even while having our first baby Emma, has allowed me to finish this endeavor with success.

Thank you Major John Comtois and the Air Force Research Laboratory (AFRL/VSSE) at Kirtland AFB, NM for the funding for my test chip. I hope that I have proven that the money was well spent to advance the state of the art.

Many thanks go to Major Charles Brothers and Dr. Ken Stevens for giving me the insight that I did not have in areas that I knew nothing about. I was truly fortunate to have experts on my committee in the areas of space electronics and asynchronous design.

Thanks also go to Major James Lott and Dr. Bruce Suter for agreeing to be on my thesis committee. Their depth of experience has helped me ensure that everything in my thesis is technically accurate and makes sense.

I must not leave out my companions in the VLSI lab. I would like to thank Captain Paul Rounsavall for his willingness to share his mastery of UNIX and for his friendship. And I must thank Greg Richardson for keeping the VLSI network alive.

Table of Contents

	Page
Acknowledgments	ii
Table of Contents	iii
List of Figures	vi
List of Tables	x
Abstract	xii
 1. Introduction	 1-1
1.1 Introduction	1-1
1.2 Problem Statement	1-1
1.3 Methodology	1-2
1.4 Overview	1-3
 2. Literature Review	 2-1
2.1 Introduction	2-1
2.2 Asynchronous Design	2-1
2.2.1 Asynchronous Versus Synchronous Design	2-2
2.2.2 Synchronous Design Flow	2-2
2.2.3 Asynchronous Design Flow	2-3
2.3 FFT and FASST Theory	2-12
2.3.1 The Fourier Transform	2-12
2.3.2 The Discrete Fourier Transform	2-12
2.3.3 The Fast Fourier Transform	2-13
2.3.4 The Suter and Stevens Fast Fourier Transform Architecture	2-15
2.4 Radiation Hardening of Electronics	2-17
2.4.1 The Need for Radiation Hardening	2-17
2.4.2 Methods of Radiation Hardening	2-21
2.5 FFT Comparison	2-24
2.5.1 General Purpose FFT Processors	2-24
2.5.2 Low-Power FFT Processors	2-25
2.5.3 High-Performance Processors	2-25
2.5.4 Previous Thesis Effort	2-26
2.5.5 Performance Comparison	2-26

2.6 Conclusion.....	2-27
3. Design Overview	3-1
3.1 Design Constraints	3-1
3.1.1 Cell Library	3-1
3.1.2 Data Type and Size	3-3
3.1.3 Project Point Size	3-4
3.2 FFT-16 Design	3-4
3.2.1 FFT-4.....	3-7
3.2.2 Complex Multiplier	3-10
3.2.3 Decimator	3-11
3.2.4 Expander.....	3-12
3.2.5 "Crossbar"	3-13
3.2.6 Constant Banks.....	3-13
3.2.7 Putting the FFT-16 Together.....	3-14
3.3 Design Conclusion	3-14
4. Design Implementation	4-1
4.1 FFT-16.....	4-1
4.2 FFT-4.....	4-2
4.2.1 Input Latches	4-5
4.2.2 Asynchronous ALU.....	4-7
4.2.3 Output Multiplexor.....	4-9
4.2.4 FFT-4 Control Units.....	4-10
4.2.5 Test Multiplexor.....	4-11
4.2.6 Final FFT-4 Design	4-11
4.3 Complex Multiplier.....	4-13
4.3.1 Multiply by One Mux.....	4-15
4.3.2 Add and Subtract ALUs	4-15
4.3.3 Radix-4 Booth Encoded Fixed-Point Multiplier.....	4-15
4.4 Constant Banks.....	4-19
4.5 Decimator	4-21
4.6 Expander.....	4-22
4.7 "Crossbar"	4-23
4.8 Reorder Register.....	4-25
4.9 Design Implementation Conclusion	4-26
5. Results	5-1
5.1 FFT-4 Test Chip	5-1
5.2 Simulation Results	5-1
6. Summary and Conclusions.....	6-1
6.1 Summary	6-1
6.2 Conclusions	6-1

6.3 Lessons Learned	6-2
6.4 Recommendations for Future Research	6-3
Appendix A. Snapshot of Fabricated FFT-4 Design.....	A-1
Appendix B. FFT-4 IC Specification Sheet	B-1
Appendix C. AFSM Descriptions and Schematics	C-1
Appendix D. Simulation Results.....	D-1
References	REF-1
Vita.....	VITA-1

List of Figures

Figure	Page
2-1. Synchronous Design Flow	2-3
2-2. Fundamental Mode Bounded Delay Applied to a Latch.....	2-4
2-3. One-Bit Adder without Completion Detection	2-6
2-4. One-bit Adder with Completion Detection	2-6
2-5. Gate Level Schematic of a Synthesized Two-Bit Johnson Counter.....	2-8
2-6. Improved Two-Bit Johnson Counter	2-9
2-7. Asynchronous Functional Block	2-10
2-8. Two-phase Model.....	2-10
2-9. Four-phase Model	2-11
2-10. Asynchronous Design Flow	2-11
2-11. FFT vs. DFT Computational Comparison.....	2-14
2-12. FASST Generic Point Size Block Diagram	2-16
2-13. I-V nMOS Curve.....	2-19
2-14. Total Dose Effects	2-20
2-15. Radiation Tolerant Layout of an Inverter.....	2-23
3-1. Inverter Size Comparison.....	3-2
3-2. Inverter Power Use Comparison	3-3
3-3. 16-bit Data Word Format	3-4
3-4. FFT-16 Vector Constant Map	3-5
3-5. FFT-16 Data Flow Block Diagram	3-6

3-6.	FFT-4 Vector Constant Map	3-8
3-7.	FFT-4 Block Diagram For 16 Add/Subtract Operations.....	3-10
3-8.	Complex Multiplier Layout.....	3-11
3-9.	Decimator Control Signals.....	3-12
3-10.	Expander Control Signals.....	3-12
3-11.	Crossbar Control Signals.....	3-13
3-12.	FFT-16 Components	3-14
4-1.	FFT-16 Top Level	4-1
4-2.	FFT-16 Components	4-2
4-3.	FFT-4 Block Diagram Using 8 ALUs.....	4-3
4-4.	FFT-4 Initial Design.....	4-4
4-5.	One-bit Latch Cell.....	4-5
4-6.	HSPICE Simulation of Register Latch Time	4-6
4-7.	LATCH16 Schematic.....	4-6
4-8.	Final ALU 1-bit Stage.....	4-8
4-9.	ALU Initialization Stage Circuit	4-9
4-10.	4x1 Mux	4-10
4-11.	Final FFT-4 Design	4-12
4-12.	Complex Multiplier Block Diagram	4-14
4-13.	Booth Multiplier Block Diagram	4-16
4-14.	34-Bit Shift Register Block Diagram.....	4-17
4-15.	Radix-4 Booth Decoder.....	4-18
4-16.	Constant Banks.....	4-21

4-17.	Decimator Gate-Level Schematic	4-22
4-18.	Expander Gate-Level Schematic	4-23
4-19.	Divide By Four Schematic	4-24
4-20.	Crossbar Schematic	4-25
4-21.	Reorder Register Schematic	4-26
5-1.	Theoretical vs. Simulation Data (Real Component)	5-3
5-2.	Theoretical vs. Simulation Data (Imaginary Component)	5-3
B-1.	FFT-4 Test Chip HSPICE Simulation of Timing.....	B-4
B-2.	FFT-4 Test Chip Power.....	B-4
D-1.	VHDL Simulation of Decimator Timing	D-1
D-2.	IRSIM Simulation of Decimator Timing	D-1
D-3.	HSPICE Simulation of Decimator Timing.....	D-1
D-4.	HSPICE Simulation of Decimator Power	D-2
D-5.	VHDL Simulation of Crossbar Timing	D-2
D-6.	IRSIM Simulation of Crossbar Timing	D-3
D-7.	HSPICE Simulation of Crossbar Timing	D-3
D-8.	HSPICE Simulation of Crossbar Power.....	D-3
D-9.	VHDL Simulation of Expander Timing.....	D-4
D-10.	IRSIM Simulation of Expander Timing.....	D-4
D-11.	HSPICE Simulation of Expander Timing	D-4
D-12.	HSPICE Simulation of Expander Power.....	D-5
D-13.	VHDL Simulation of Reorder Register Timing	D-5
D-14.	IRSIM Simulation of Reorder Register Timing	D-5

D-15. HSPICE Simulation of Reorder Register Timing	D-6
D-16. HSPICE Simulation of Reorder Register Power.....	D-6
D-17. VHDL Simulation of FFT-4 Timing	D-6
D-18. IRSIM Simulation of FFT-4 Timing	D-7
D-19. HSPICE Simulation of FFT-4 Timing	D-7
D-20. HSPICE Simulation of FFT-4 Power.....	D-7
D-21. VHDL Simulation of Complex Multiplier Timing	D-8
D-22. IRSIM Simulation of Complex Multiplier Timing	D-8
D-23. HSPICE Simulation of Complex Multiplier Timing.....	D-8
D-24. HSPICE Simulation of Complex Multiplier Power	D-9
D-25. VHDL Simulation of FFT-16 Timing	D-9
D-26. IRSIM Simulation of FFT-16 Timing	D-9
D-27. FFT-16 VHDL Simulation of Impulse Response	D-10
D-28. FFT-16 IRSIM Simulation of Impulse Response	D-10

List of Tables

Table	Page
2-1. 3D State Table of a Two-Bit Johnson Counter	2-8
2-2. A Comparison of FFT Processors	2-26
4-1. FFT-4 Area Comparison	4-4
4-2. ALU Initialization Truth Table	4-9
4-3. Radix-4 Booth Algorithm.....	4-17
4-4. Constant Bank One Values	4-20
4-5. Constant Bank Two Values.....	4-20
4-6. Constant Bank Three Values.....	4-20
5-1. Impulse Function Input Sequence	5-2
5-2. Matlab vs. Simulation Results.....	5-2
5-3. FFT-16 Design and Simulation Results	5-4
5-4. Final Comparison of FFT Processors.....	5-6
B-1. FFT-4 Test Chip Specifications	B-1
B-2. Fabricated FFT-4 Pin List	B-2
B-3. Test Signals	B-3
C-1. FFT-4 CONTROLIN AFSM.....	C-1
C-2. FFT-4 CONTROLGETREG AFSM Description	C-2
C-3. FFT-4 CONTROLOUTA AFSM Description	C-3
C-4. FFT-4 CONTROLOUTB AFSM Description	C-4
C-5. Booth Multiplier CONTROLMULT AFSM Description	C-5

C-6.	Booth Multiplier CONTROLCALC AFSM Description.....	C-6
C-7.	Booth Multiplier CONTROLBOOTH AFSM Description.....	C-7

Abstract

A second-generation fully asynchronous Fast Fourier Transform (FFT) processor for space applications is developed in this thesis. A high-performance patented FFT architecture invented by Suter and Stevens was used as the basis for a 16-point FFT (FFT-16) processor design. A brief derivation of the architecture, the asynchronous design methodologies used and space-based integrated circuit issues are presented. The Synopsys VLSI CAD system and a radiation tolerant design library developed by the Air Force Research Laboratory were used to implement the design. A critical building block of the FFT-16, the FFT-4, was fabricated as a cost-effective method to validate the cell library and the applied asynchronous design methodologies before larger point sizes are fabricated.

Results from high-fidelity simulations show that the FFT-16 design has an efficiency of 28 nJ/Unit-Transform and has a worst case throughput of 760 ns. Extrapolating these results to an FFT-1024 gives an estimated efficiency of 120 nJ/Unit-Transform and worst case throughput of 2 μ s. These results demonstrate that current space-based FFT processors can be replaced with a design that improves performance and efficiency by two orders of magnitude.

AN IMPROVED ASYNCHRONOUS IMPLEMENTATION OF A FAST FOURIER TRANSFORM ARCHITECTURE FOR SPACE APPLICATIONS

1. Introduction

1.1 Introduction

The goal of this research is to investigate, design and implement an asynchronous very large scale integrated (VLSI) circuit targeted for space applications that calculates a Fast Fourier Transform (FFT) using the Suter and Stevens architecture [1]. The research presented in this thesis is the continuation of work performed in a previous thesis [2]. An improved design of a 16-point FFT (FFT-16), from initial concepts to the simulation test results is presented. A subset of the design (an FFT-4) was fabricated using the Hewlett-Packard 0.5 μm commercial process for future analysis.

1.2 Problem Statement

There is an identified need for a fast, low power FFT processor for space applications. Theoretically, the implementation of the Suter and Stevens architecture in an asynchronous fashion should result in an extremely fast, low power FFT processor.

Such a design can also be used in the space environment by replacing the standard VLSI cells with radiation tolerant cells.

The Suter and Stevens architecture inherently lends itself to an asynchronous implementation. However, current VLSI design tools are not capable of asynchronous circuit synthesis. Consequently, a large portion of this research covers the asynchronous design methodology. Asynchronous design implies that the global clocking strategy used in synchronous design is removed and replaced with a self-timing scheme, which lowers the energy requirement of the circuit [3]. The FASST (Fully Asynchronous Suter Stevens Transform) acronym used throughout this thesis refers to the asynchronous implementation of the Suter and Stevens architecture.

The space application requirement is met by using a VLSI design library, developed jointly by the Air Force Research Laboratory (AFRL) and Mission Research Corporation (MRC) [4]. This unique library enables a circuit design for space application to be fabricated in a commercial foundry. The tradeoff for using this library is that both the overall die area and energy requirement is increased for the design.

1.3 Methodology

The first step in a VLSI design is to define the top-level function of the circuit. Then, initial design constraints are chosen, including the CMOS technology and the data word format. Goals are established for area, power and performance. The architecture of the design is then selected. The next step is to break down the architecture into manageable blocks, with well-defined interfaces and specifications. Each block is

developed from the behavioral level to the physical layout and simulated at each level for verification. When all blocks are complete, they are tested together to verify top-level operation. Circuit extractions from the physical layout are used to test the circuit in a high fidelity simulation in order to evaluate the function, performance and efficiency of the design. The design and verification process is repeated until the established design goals are met. Once the design is fabricated, it can be tested and compared to the simulation results [5].

1.4 Overview

This thesis is organized into six chapters. The chapters are organized in a logical manner beginning with the problem statement and concluding with results. Chapter One introduces the problem, the design methodology used and includes an overview of the thesis.

Chapter Two provides an overview of asynchronous circuit design methodologies, FFT theory and the radiation hardening of electronics. The chapter concludes with a presentation of other work related to this research area. The chapter highlights the significance of the problem and gives the motivation for this thesis.

Chapter Three presents an overview of the FFT-16 design. The theory presented in Chapter Two is applied to the development of a working solution. The function of the top-level design and the major components are discussed.

Chapter Four is a presentation of the design implementation of each functional block. Each block introduced in Chapter Three is revisited in detail. The final design of each block, as well as other possible designs, are presented.

Chapter Five is a presentation and analysis of the simulation results. Results are given at each level of design for the individual components as well as the results of the top-level design.

Chapter Six concludes the thesis by comparing the results of this research with the research efforts presented in Chapter Two. Lessons learned during the design process are presented. Recommendations for future work in this area are also given.

2. *Literature Review*

2.1 *Introduction*

The purpose of this chapter is to present applicable research in the subject areas of asynchronous design, FFT theory, and radiation hardened electronics. A literature search failed to identify a single design or any research, which combines these areas other than a previous student's thesis on the same topic [2][6]. There are a few examples of designs, which overlap two of these areas [7]. The chapter concludes with a presentation of other FFT processors that are comparable to this effort. A table is presented in Section 2.5 summarizing the performance of these designs.

2.2 *Asynchronous Design*

Asynchronous circuit design is not a new concept in electronic circuit design. In fact, asynchronous circuits have been used since the 1950's but have not been widely adopted by modern industry [8]. Asynchronous circuits have the potential to out-perform synchronous circuits. However, the tools and support community are still not as mature as those used in the mainstream development of synchronous circuits. The following sections contrast the synchronous and asynchronous design methodologies [3][9].

2.2.1 Asynchronous Versus Synchronous Design

Synchronous design implies that a global clock is used to synchronize the exchange of data between components in a design. Logic blocks are typically surrounded by latches, which save the state of the block during each clock cycle. The clock rate is defined by the critical path through the system [5].

Asynchronous design removes the global clock and replaces it with a self-timed protocol. Interconnected blocks communicate and exchange data with a sequence of handshakes [3].

2.2.2 Synchronous Design Flow

Traditional synchronous VLSI circuits are designed using modern synthesis tools. A hardware description language (HDL) is used to describe the behavior of the circuit. The Very High Speed Integrated Circuit Hardware Description Language (VHDL) is the DoD standard and consequently was used in this effort [10].

Typically, a high level behavioral description of the circuit under development is written in VHDL. The behavioral VHDL is then translated into structural VHDL with a tool such as the Synopsys Design Analyzer [11]. This essentially translates the design from a logical algorithmic behavior to a realizable gate level structural design. Once the structural design is verified, a layout netlist, which describes the component connections, is generated from a tool such as the Synopsys Graphical Environment [12]. The circuit netlist is then used with a standard cell library to produce a final layout with a place and route tool such as the Lager Octtools [13]. Figure 2-1 summarizes the typical synchronous design flow.

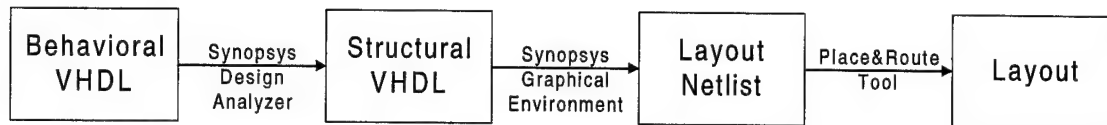


Figure 2-1. Synchronous Design Flow

2.2.3 Asynchronous Design Flow

While an established design flow exists for synchronous circuits, the same is not true for asynchronous design. No suite of design tools exists that allows an easy flow from behavioral VHDL to an automated layout. Similar to synchronous design, the asynchronous design process begins with a behavioral VHDL description. However, the automated process ends here and is replaced with a combination of manual design and partial automatic synthesis to arrive at a structural VHDL description. The design flow continues as normal once the structural VHDL is validated. The methods used in the design phase between behavioral and structural VHDL are described in this section.

The components designed in this effort fell into several design categories. The fundamental mode bounded delay methodology is used for blocks with relatively fixed completion times. The delay insensitive design methodology applies to functional blocks with widely varying completion times. Burst mode design methodology applies to components that serve as controllers or asynchronous finite state machines (AFSMs). Finally, the speed independent model specifies the handshaking protocols between major functional blocks. These issues are presented in the following sections [3].

2.2.3.1 Fundamental Mode Bounded Delay Methodology

The fundamental mode bounded delay methodology was used for functional blocks that had little variation in completion time [14]. This methodology assumes that

the delay time through a functional block is known and constant. Worst-case delays are used similar to a clocked circuit. The best example of this type of functional block is a latch. The easiest way to determine when data is fully latched is through a delay element. The delay element should have a slightly greater delay than the completion time of the logic block [3].

Difficulty arises in synthesizing this structure since timing information cannot be synthesized from behavioral VHDL. The delay element must be described at the structural level. The length of the delay element is determined through a layout-level simulation of the logic block. The results of the simulation are then back annotated into the structural VHDL design. Figure 2-2 shows a delay element used to model the latch completion time. An acknowledge (ACK) signal is asserted when the data is latched after the request (REQ) is generated.

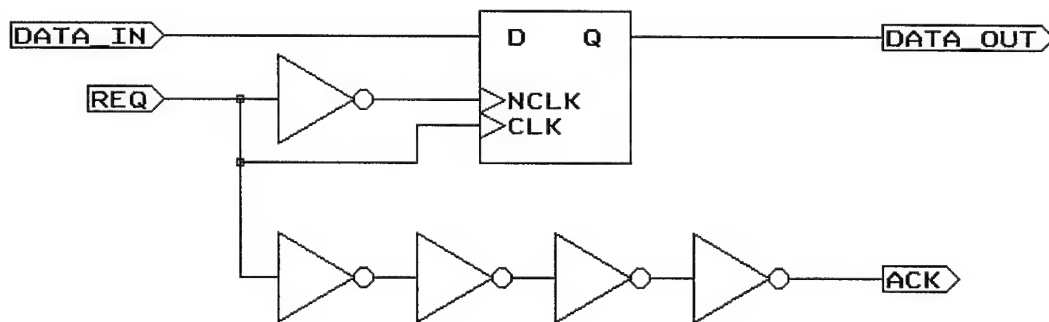


Figure 2-2. Fundamental Mode Bounded Delay Applied to a Latch

2.2.3.2 Delay Insensitive Methodology

A delay element is not suitable for functional blocks with widely varying completion times, since the benefit of an average delay throughput is not realized.

Additional logic can be added to this type of block to detect when its execution is complete [3].

VLSI synthesis tools do not have the capability to generate the completion detection circuit for a particular functional block. For example, they do not know how to synthesize the completion detection circuit for an adder. Figure 2-3 shows a typical one-bit adder without completion detection.

A dual-rail adder scheme similar to the Manchester adder can be used to implement completion detection, as shown in Figure 2-4 [15]. The dual rail adder works on the principle that each stage will have either a carry out (COUT) or no carry out (NOCOUT) condition based on the inputs to the stage. Adding 0 and 0 will never result in a carry out, even if there is a carry in. Likewise, adding 1 and 1 will always result in a carry out, even if there is a carry in of 0. Therefore, the carry condition in these cases can be determined by the data to be summed alone and gives an early completion detection. Adding a 0 and 1 or 1 and 0 may or may not have a carry out depending on the carry in condition. In this case, the stage must wait for either a carry in (CIN) or no carry in (NOCIN) value. The end result is the completion detection circuit simply becomes the NOR of the COUT and NOCOUT values. Whenever one of these conditions exist, it indicates that all input values necessary for evaluating the sum are present and DONE is asserted.

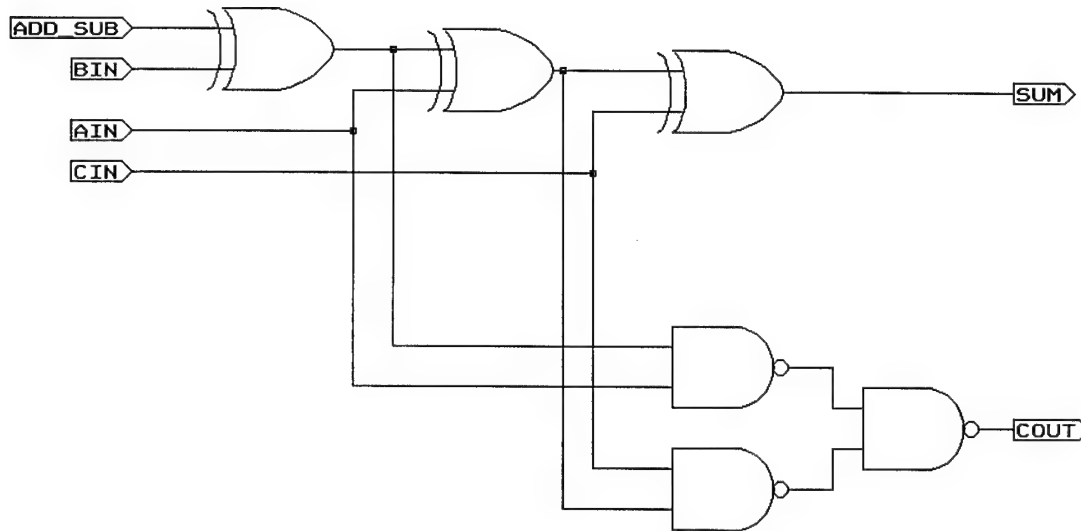


Figure 2-3. One-Bit Adder without Completion Detection

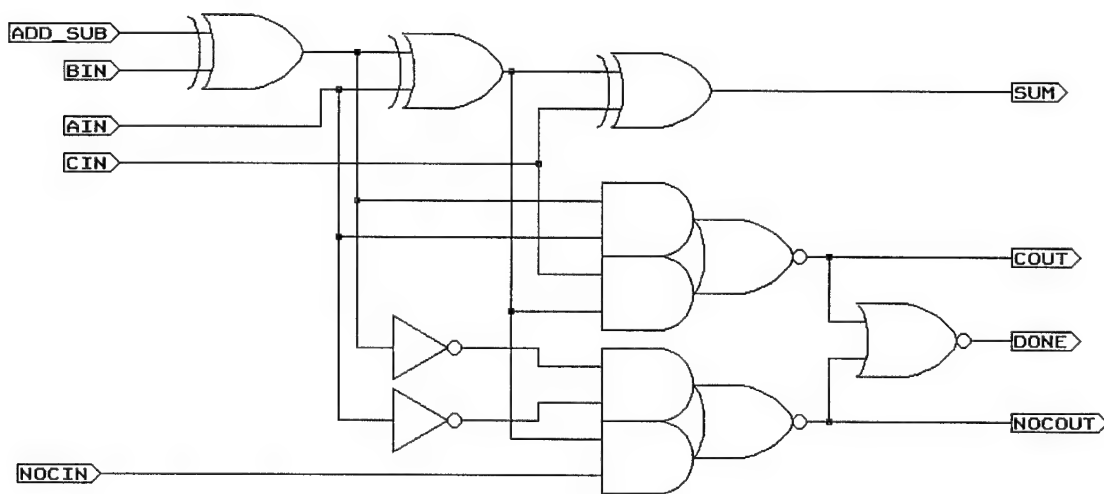


Figure 2-4. One-bit Adder with Completion Detection

2.2.3.3 Burst Mode Methodology

The burst mode design methodology is used to design asynchronous controllers or finite state machines. Synchronous finite state machines are easily synthesized by using latches, flip-flops and clock circuitry. Asynchronous controllers or AFSMs must be

synthesized using a specialized design tool. Unfortunately, no commercial tools exist with this capability [3].

AFSMs can be designed by hand, but is a very tedious and error-prone process. Tools have been developed by universities and corporate laboratories that automatically synthesize AFSMs. The Most Excellent Asynchronous Tool (MEAT) is an early example of such a tool [16]. A more recently developed tool called 3D took the basic principles of MEAT and further refined them. The 3D tool was used in the design phase of this effort because it is kept up to date and maintained by UC San Diego [17].

A state table of entry and exit conditions for the state machine is provided to 3D by the user. An example state table is shown in Table 2-1 for a Johnson counter (00→01→11→10). 3D converts the state table to positive logic equations. These equations are then manually converted into behavioral VHDL. The Synopsys Design Analyzer (with structuring and Boolean optimization disabled) is used to convert the positive logic behavioral VHDL into negative logic structural VHDL. After the structural VHDL is generated, reset circuitry and corrections for fanout are added manually to the controller circuit. The final two-bit Johnson counter circuit is shown in Figure 2-5, which includes the reset circuit. Once the structural VHDL is complete, it is tested using a VHDL simulator.

Depending on the complexity of the AFSM, 3D may not be able to synthesize the controller. The controller must then be broken down using Shannon decomposition and resynthesized. Once an AFSM has been synthesized and validated in VHDL, it can be used in a physical layout.

Table 2-1. 3D State Table of a Two-Bit Johnson Counter

Present State	Next State	Entry Conditions	Exit Conditions
0	1	COUNT+	
1	2	COUNT-	BIT0+
2	3	COUNT+	
3	4	COUNT-	BIT1+
4	5	COUNT+	
5	6	COUNT-	BIT0-
6	7	COUNT+	
7	0	COUNT-	BIT1-

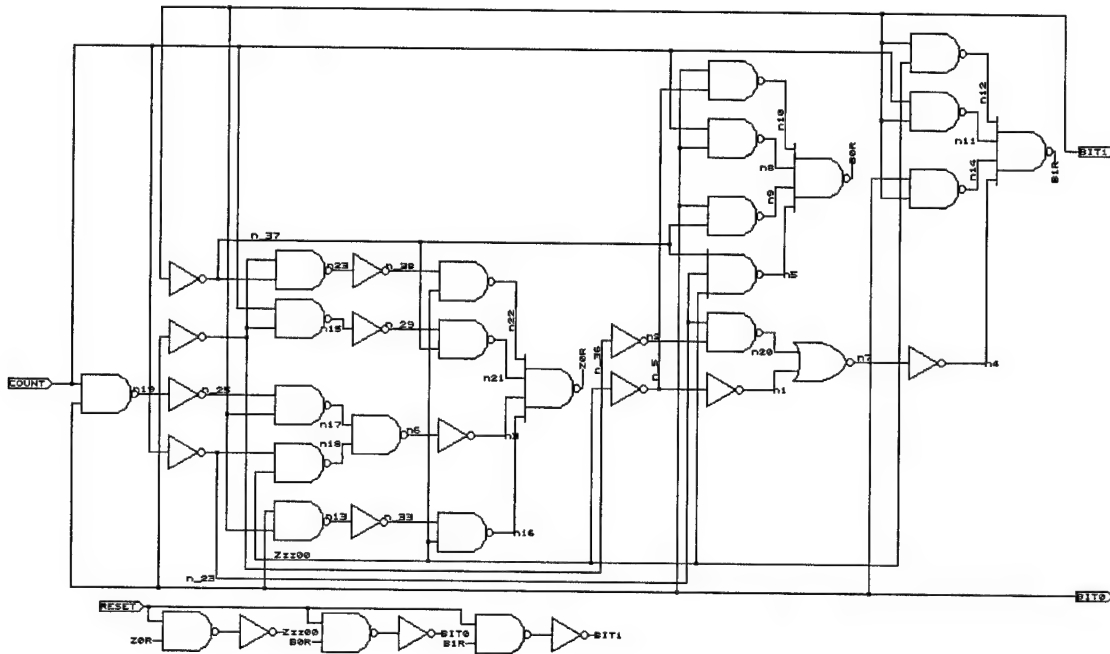


Figure 2-5. Gate Level Schematic of a Synthesized Two-Bit Johnson Counter

The two-bit Johnson counter example is used to illustrate how asynchronous synthesis tools work, but it highlights how automated AFSM synthesis does not always produce the optimal solution [9]. A better implementation of the two-bit Johnson counter is accomplished by using two D-registers, as shown in Figure 2-6. This type of counter is

used throughout the design. The Johnson counter was selected due to the fact that it changes only one bit each clock cycle, thus avoiding possible data hazards.

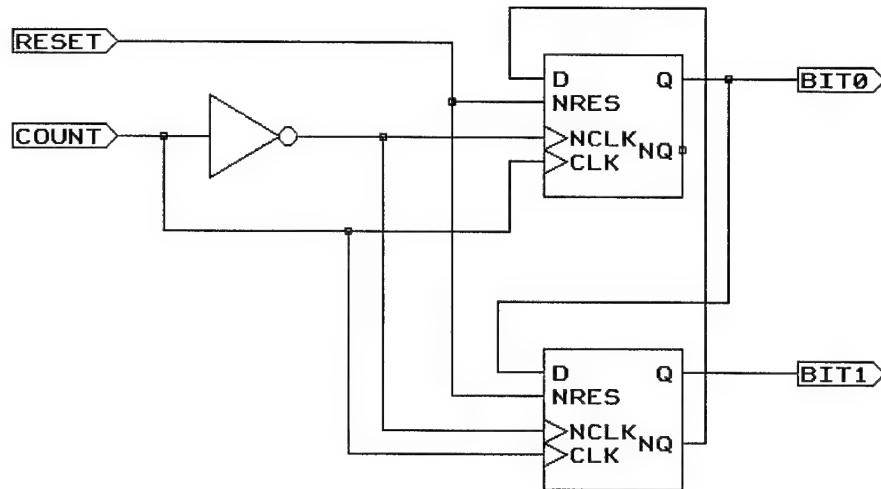


Figure 2-6. Improved Two-Bit Johnson Counter

2.2.3.4 Speed Independent Methodology

Functional blocks in an asynchronous design must have a standard handshaking protocol in order to be compatible with other blocks. A generic functional block in an asynchronous design is shown in Figure 2-7. The input and output signal names shown here are used throughout this research effort. The REQIN signal represents the external request to the block to input new data. The ACKIN signal is asserted when the new input data is fully latched or accepted. The REQOUT signal represents the request of the functional block to send processed data out. The ACKOUT signal is the external acknowledgement from the next block that the processed data was latched or accepted.

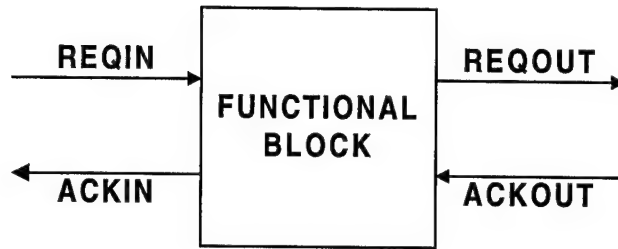


Figure 2-7. Asynchronous Functional Block

The speed independent methodology describes two standards for handshaking between connecting blocks. It does not assume any pre-defined delays but relies on a set of handshaking signals between the blocks. The two-phase model is illustrated in Figure 2-8. It is a scheme that senses signal transitions to complete the handshake cycle. The first exchange is signaled by a low to high transition on REQ (1). ACK (2) responds by acknowledging the request. The second cycle uses the complementary set of transitions to complete the cycle [3].

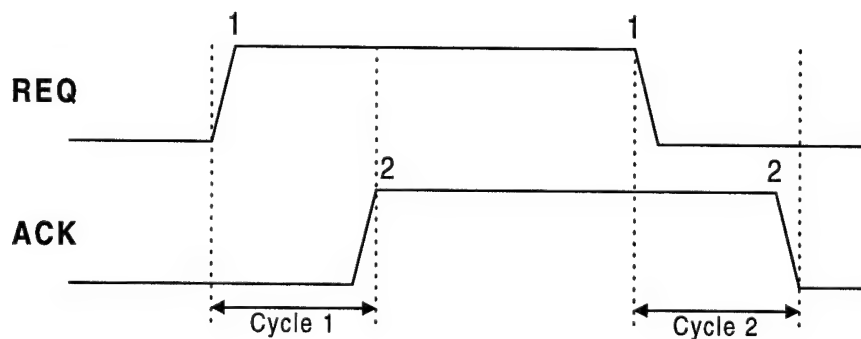


Figure 2-8. Two-phase Model

The four-phase model is illustrated in Figure 2-9. It has a four-cycle handshake for each data exchange. Although the four-phase model appears to be more difficult to implement, its detection circuit is actually smaller than the two-phase model [3]. The four-phase model is the primary interface standard used throughout this design.

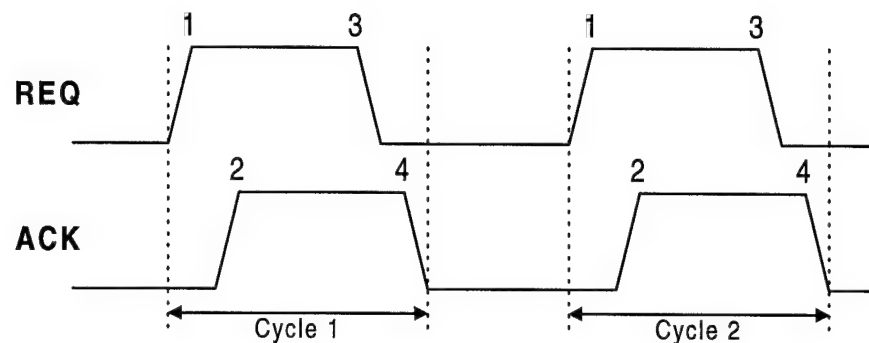


Figure 2-9. Four-phase Model

2.2.3.5 Asynchronous Design Flow Summary

Figure 2-10 illustrates the asynchronous design flow used in this research. Comparing Figure 2-10 with Figure 2-1 highlights the additional manual intervention necessary to arrive at a complete asynchronous design compared to a synchronous design.

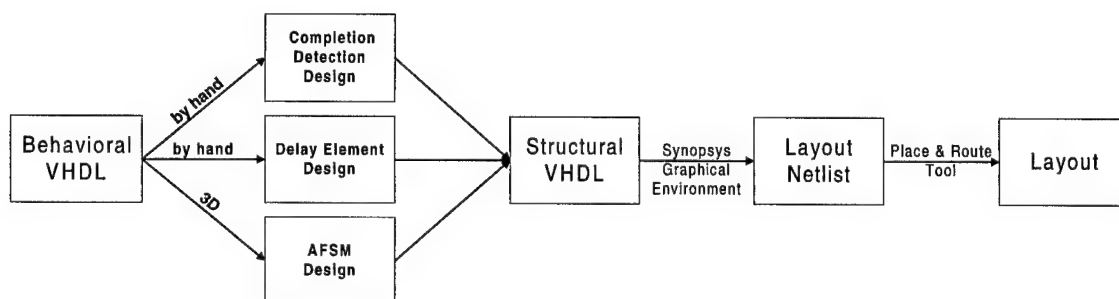


Figure 2-10. Asynchronous Design Flow

Research using the asynchronous design methodology presented additional challenges over those present in synchronous design. Prototype design tools and manual circuit design involving an iterative trial and error process were used to bridge the gaps between the asynchronous tools and the VLSI design tools.

2.3 *FFT and FASST Theory*

Computing an FFT is a very efficient way to digitally convert a time domain signal into a frequency domain signal. A short explanation of the FFT and FASST are provided in this section.

2.3.1 *The Fourier Transform*

The Fourier Transform is a mathematical operation that is used to convert data in the time domain to the frequency domain. The basic equation is shown in Equation 2-1 where $x(t)$ is the time domain signal, $X(f)$ is the transformed frequency domain component, and the exponential represents the Fourier series components [18].

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt \quad (2-1)$$

To solve the Fourier Transform using a computer, the input data and calculations must be broken into finite segments of a given size and processed using a slightly different formula. This modified formula is called the Discrete Fourier Transform (DFT) and is introduced in the next section.

2.3.2 *The Discrete Fourier Transform*

Once can apply numerical integration to Equation 2-1 to create a problem that is more amenable to implementation on a computer as a trapezoidal formula since $x(t_i) = x(t_n)$ as shown in Equation 2-2.

$$X(f_k) = \sum_{i=0}^{N-1} x(t_i)e^{-j2\pi f_k t_i}, k = 0, \dots, N-1 \quad (2-2)$$

Typically, one will analyze a problem where there are N data points of the function, which are assumed to be represented by no more than N different sinusoids. Equation 2-2 reveals a complexity of N where each operation requires a multiplication. Thus we can approximate the continuous Fourier transform using a discrete representation of the transform by letting $t_i = n\delta t$ and $f_k = m\delta f$ where $\frac{1}{\delta f} = N\delta t$. With these substitutions, the discrete Fourier transform can be represented as in Equation 2-3.

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j \frac{2\pi n m}{N}} \quad (2-3)$$

N represents the number of samples of the finite sequence (which is commonly referred to as the *point size* of a DFT), $x(n)$ represents the time domain values of the sequence, $X(m)$ represents the frequency domain components of the Fourier Transform of $x(n)$. There are N complex multiply operations required to solve the equation, which translates to $4N$ real multiply operations on a computer (this makes a total of $N \times 4N$ or $4N^2$ operations which is expressed as $O(N^2)$). By taking advantage of complex conjugate symmetry, the DFT can be solved in less than $O(N^2)$ operations. This concept is the premise of the Fast Fourier Transform (FFT) [18].

2.3.3 The Fast Fourier Transform

The FFT was developed in the 1960's when signal processing was becoming an interesting research tool. Limited computational resources sometimes prohibited using the DFT to evaluate large point sizes. An algorithm, which simplified the computation of

the DFT, was developed and was later called the Fast Fourier Transform [19]. This transform is briefly described in this section.

The first step in simplifying the DFT is to use the substitution $W_N = e^{-j\frac{2\pi}{N}}$. The DFT can be expressed as shown in Equation 2-4.

$$X(m) = \sum_{n=0}^{N-1} x(n)W_N^{nm} \quad (2-4)$$

By taking advantage of the complex conjugate symmetry of W_N , the number of overall computations is reduced from $O(N^2)$ down to $O(N\log_2 N)$ which classifies the algorithm as a “fast” Fourier Transform or FFT. This is an extremely advantageous property as such a reduction before implementing an algorithm in software or hardware will realize a substantial performance increase. The comparison of the number of multiplication operations required by the FFT and DFT is clearly shown in Figure 2-11 [18].

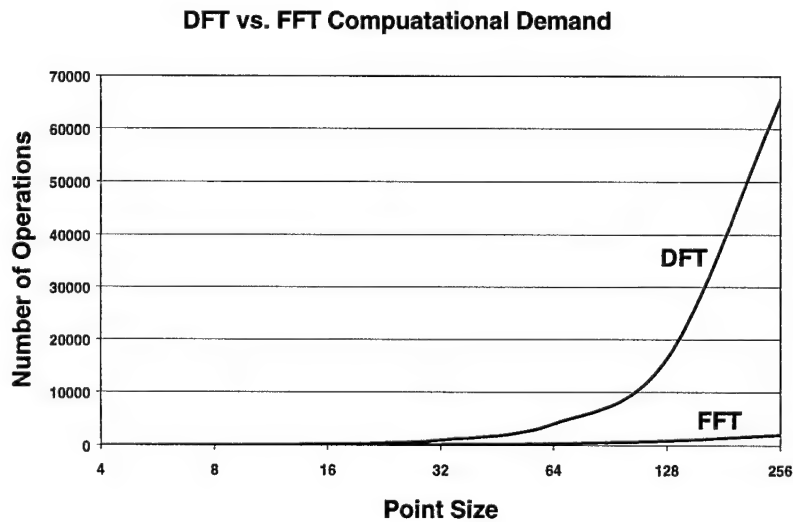


Figure 2-11. FFT vs. DFT Computational Comparison

2.3.4 The Suter and Stevens Fast Fourier Transform Architecture

The Suter and Stevens FFT architecture is pipelined, extremely local and eliminates the need for shared memory. It utilizes a small number of logic blocks that are replicated throughout the architecture and operate in parallel [20].

A simple derivation is given here of the architecture. Referring to Equation 2-4, the substitution $N = N_1 N_2$ is made. Using the division theorem for integers, $m = m_2 N_1 + m_1$ and $n = n_1 N_2 + n_2$ where $m_1, n_1 = 0, 1, \dots, N_1 - 1$ and $m_2, n_2 = 0, 1, \dots, N_2 - 1$. The polyphase components [21] of $x(n)$ are defined as $x_k(n) = x(M_n + k)$, $k = 0, \dots, M - 1$. The FFT can be broken into interdependent equivalent classes of calculations, or similar types of calculations, by letting $X_{m_1}(m_2) = X(m_2 N_1 + m_1)$ and $x_{n_2}(n_1) = x(n_1 N_2 + n_2)$. This polyphase notation is used to represent Equation 2-3 as Equation 2-5

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j \frac{2\pi(m_2 N_1 + m_1)(n_1 N_2 + n_2)}{N}} \quad (2-5)$$

which is equivalent to Equation 2-6.

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j \frac{2\pi m_2 N_1 n_1 N_2}{N}} e^{-j \frac{2\pi m_2 N_1 n_2}{N}} e^{-j \frac{2\pi m_1 n_1 N_2}{N}} e^{-j \frac{2\pi m_1 n_2}{N}} \quad (2-6)$$

Equation 2-7 is the result of factoring and simplifying the first exponential term to unity.

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \left[e^{-j \frac{2\pi m_1 n_2}{N}} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) e^{-j \frac{2\pi m_1 n_1}{N_1}} \right] e^{-j \frac{2\pi m_2 n_2}{N_2}} \quad (2-7)$$

Using the W_N substitution, Equation 2-8 mathematically describes the FASST architecture.

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \left[W_N^{m_1 n_2} \sum_{n_1=0}^{N_1} x_{n_2}(n_1) W_{N_1}^{m_1 n_1} \right] W_{N_2}^{m_2 n_2} \quad (2-8)$$

Finally, Equation 2-8 is expressed in hardware by Figure 2-12.

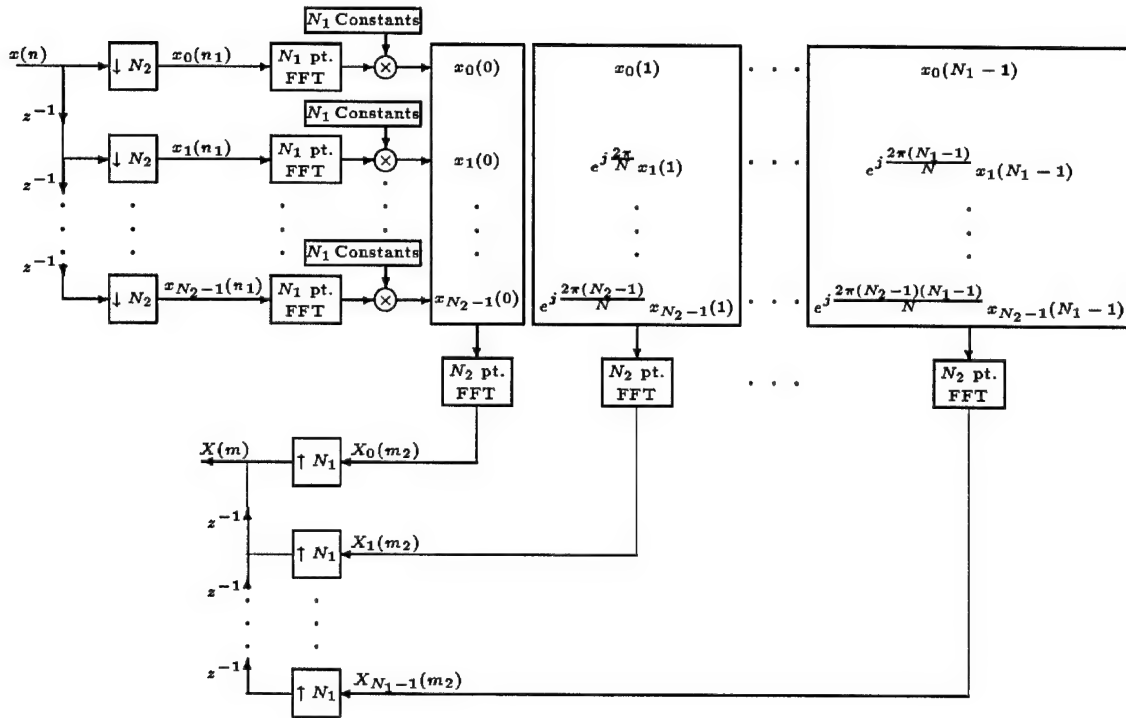


Figure 2-12. FASST Generic Point Size Block Diagram [20]

Figure 2-12 implies that there are several basic components of the architecture. The $\downarrow N_2$ blocks are decimators, which break down the N input values into N_1 concurrent streams at a frequency of $1/N_1$. The N_1 and N_2 FFT blocks are the foundational FFT components. The \otimes components are the complex multipliers. The N_1 constants store the appropriate constant values depending on the point size. The large boxes are routing areas for post-multiplied values before they are processed by the N_2 FFT. The final component is an expander, which is shown as $\uparrow N_1$ in the figure. The expanders compose

the output values of the N_2 FFT into the correct output order and reduce the frequency by a factor of N_2 . An overview of these components is provided in Chapter Three and a detailed presentation of how they were implemented is presented in Chapter Four.

2.4 *Radiation Hardening of Electronics*

The term *radiation hardening* originated from a class of military electronics that had to operate through and survive the most severe radiation environments. Although many applications require protection from radiation, they do not require the highest level of protection attainable. Cost factors dictate that application specific electronics only be *radiation tolerant*. A variety of methods can be used to protect a circuit from the effects of radiation and several of them are described in this section [22].

The need has been demonstrated for a high performance FFT processor designed for the space environment and has driven the requirement to make the product of this research radiation tolerant. The space environment introduces many hazards not present on earth. Additional design measures must be taken to prevent these hazards from impacting the operation of this design.

2.4.1 *The Need for Radiation Hardening*

The United States Air Force has an interest in circuits that are able to perform in a radiation environment. Circuits used in space must have some degree of radiation tolerance. This effort explores two categories of radiation exposure: long term total ionizing dose and short lived single event effects [22].

2.4.1.1 *Total Ionizing Dose*

Complementary metal oxide semiconductor (CMOS) circuits account for a significant majority of the world's electronic circuits [5]. They degrade in a radiation environment due to the total accumulated dose of radiation. This degradation is seen as a negative shift in the transistor threshold voltage and decrease in gain. With enough voltage threshold shift, the circuit will start consuming power even when not switching. The decrease in gain causes the transistors to become harder to switch. After extended exposure to radiation, the circuit will cease to function [23].

The main source of degradation comes from the interaction of ionizing radiation with the gate and field oxides (SiO_2) in the device structure. The gate oxide is a thin high-quality oxide used to insulate the gate contact from the transistor channel. The field oxide is a thick low-quality oxide used to isolate metal traces from one another [22].

Ionizing radiation causes the formation of electron-hole pairs in the gate oxide. Electrons have a much higher mobility than holes in SiO_2 and are attracted to and swept out of the gate in a nMOS transistor. The holes become trapped and migrate toward the transistor channel. This results in the eventual buildup of positive charge above the transistor channel and acts like the charge that is present when voltage is applied at the gate. As more charge is trapped, the voltage threshold of the nMOS transistor becomes more negative, which means it becomes easier to turn on. With enough shift in threshold voltage, the transistor will be turned on without a gate voltage applied. Conversely, a pMOS transistor becomes more difficult to turn on. Figure 2-13 shows how the gate voltage versus drain current curve changes resulting from exposure to radiation in a nMOS transistor [23].

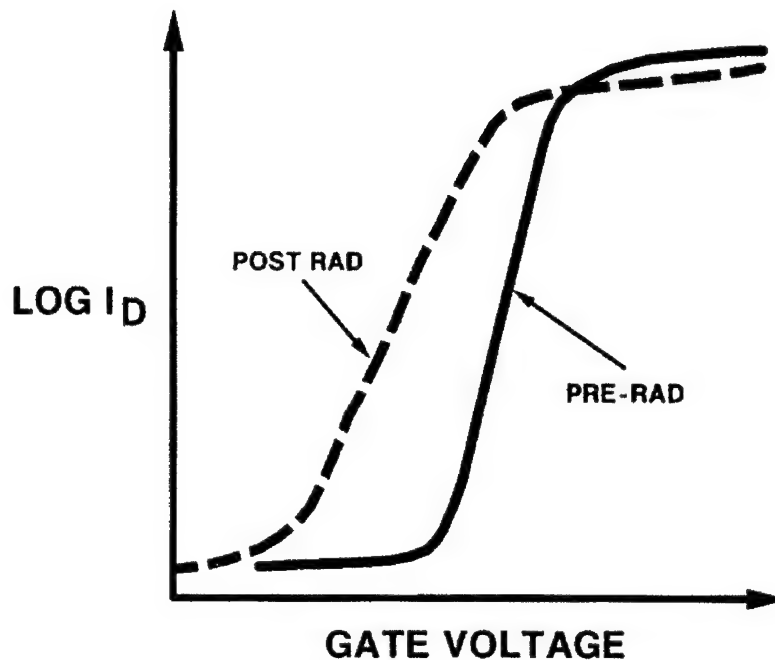


Figure 2-13. I-V nMOS Curve [23]

The field oxide also traps charge due to ionizing radiation. The trapped positive charge along the edges of the nMOS transistor creates a leakage channel. Leakage paths can also form between transistors through the field oxide. This constant leakage contributes to increased power consumption [22].

Figure 2-14 illustrates how a circuit exposed to a radiation environment slowly increases power consumption and reduces the operating frequency. Eventually, the circuit will cease functioning when the power required by the degraded electronics exceeds the output capability of the power supply. Premature failure can also occur when the output voltage swing of the transistors becomes insufficient to drive successive stages or when the timing is degraded to the point where the circuit does not operate properly [23]. An important thing to note about an asynchronous space design is that it will

automatically adjust its operating frequency which can potentially extend the life of the circuit.

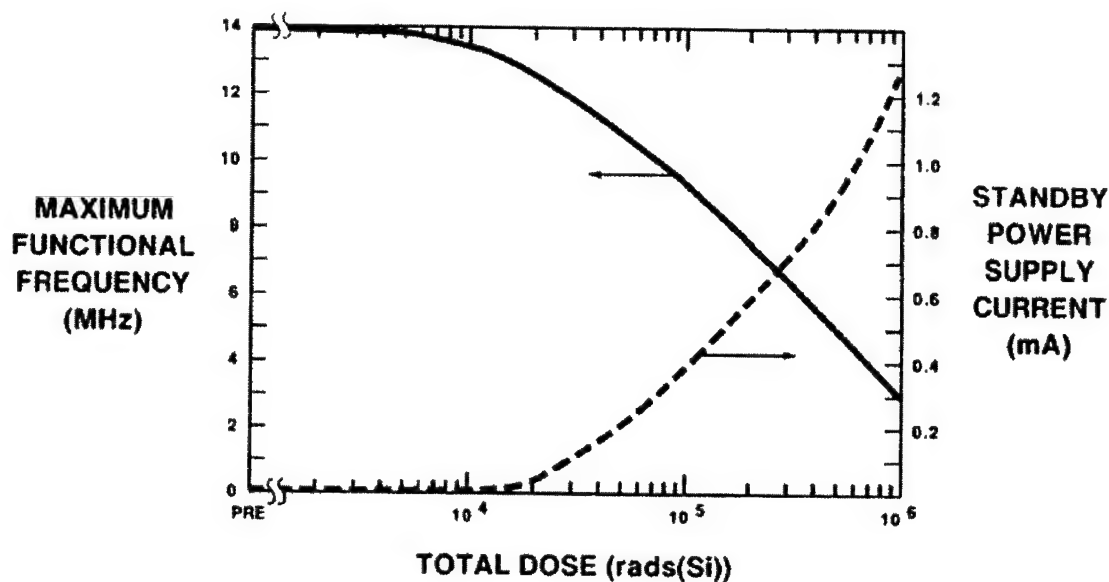


Figure 2-14. Total Dose Effects [23]

2.4.1.2 Single Event Effects

When a high-energy particle passes through a circuit and causes a disruption in circuit operation, it is classified as a single event effect (SEE). For example, a proton or ion passing through a latch could change the value of a stored bit. This event is called a single event upset (SEU) [24].

Protons and high-energy heavy ions typically cause SEUs. Space vehicles passing through the South Atlantic anomaly, where there is a high concentration of protons can experience SEU activity in that region. These particles create a temporary presence of an abundance of free carriers in the transistor channel region. The free carriers in effect turn the channel on [24].

If a channel is turned on in a combinational logic circuit, the effect is seen as a spike in the output data and usually does not affect system operation. However, if a channel is turned on that is part of a memory structure, such as a latch, it can upset the state of the latch. Upset can only occur if enough carriers are present in the transistor channel to turn it on strongly enough to change the state of the latch. SEU can be corrected by refreshing memory locations on a periodic basis [24].

Another effect seen in CMOS is single event latchup (SEL). SEL describes the phenomenon that occurs when inactive parasitic transistor regions (pnpn structure) are turned on by a high-energy particle. These pnpn regions are formed in CMOS layouts due to the close placement of nMOS and pMOS transistors and have the characteristics of a silicon controlled rectifier (SCR). If a particle with enough energy passes through the controlling pn junction of the SCR, it can switch the SCR on. The only way to turn the SCR off is with a power cycle [24].

2.4.2 Methods of Radiation Hardening

Radiation hardening was first used by the military to ensure that critical systems would be operative during a nuclear war [22]. However, this technology has become applicable to commercial systems as more satellites are being placed in orbits with elevated radiation levels. There are many ways of achieving radiation hardness. Each can be used individually or combined to achieve the highest level of radiation tolerance. Shielding, fabrication process, and design layout techniques are typical methods used to achieve radiation hardness [22].

2.4.2.1 Radiation Hardening through Shielding

The most intuitive way to protect a circuit is by enclosing it in a metal box thick enough to shield against all radiation. This is impractical, especially if it is to be used on a spacecraft where weight is a concern. Shielding usually involves surrounding electronics with 200-300 mils of aluminum, which is usually provided by the satellite body. This shielding will block out low-energy particles, but does little to stop high-energy particles. Additional protective measures to ensure radiation tolerance are discussed in the next sections [24].

2.4.2.2 Radiation Hardening through Fabrication

The most sophisticated but costly way to harden a circuit is to alter the fabrication process. The thickness and growth method of the gate oxide is altered. Thinner gate oxides are more resistant to total ionizing dose. High quality oxides also increase total ionizing dose resistance. Both of these methods work by reducing the amount of charge trapped in the oxide. These methods drive up the cost, as thin oxides require precise controls and high quality oxides require more time to grow [22].

Another fabrication technique used to increase radiation hardness is to grow the transistor structures on a high quality insulating material. This method reduces the frequency of single event effects. By growing devices on an insulator, the parasitic transistor region is eliminated, thus preventing SEL. SEU is also reduced, as line charge formation from an ion strike is minimized [24].

2.4.2.3 Radiation Hardening through Layout

Another method for increasing radiation tolerance of a VLSI circuit is to change the design rules by which the circuit is laid out. Usually this results in a larger area due to high drive strength devices, implying a loss of power efficiency [22].

Radiation hardening through layout is the only viable method of producing a radiation tolerant design for a thesis, as the design can be fabricated on an inexpensive commercial process line. The only additional cost comes from the increased area requirement. The gate and pad library designed jointly by AFRL and MRC used in this effort achieves maximum radiation tolerance from a commercially fabricated circuit. The library is designed for fabrication using the HP 0.5 μm process [4].

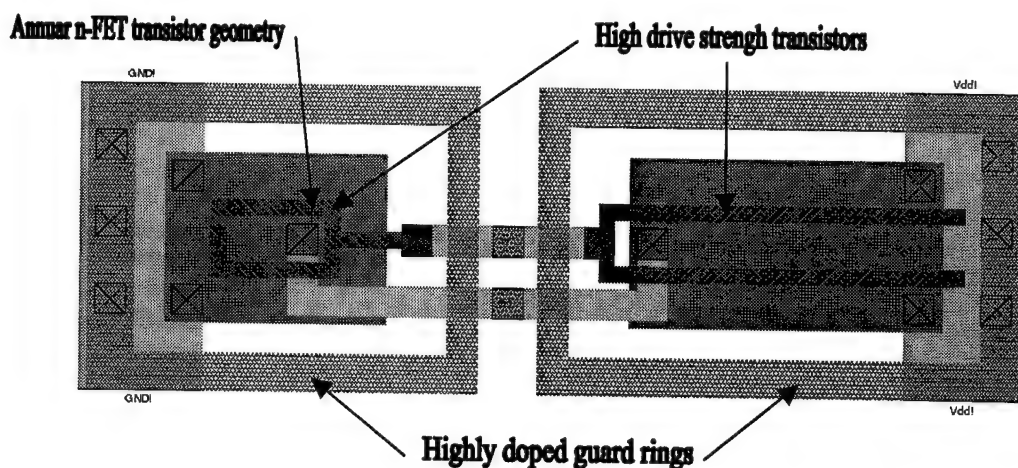


Figure 2-15. Radiation Tolerant Layout of an Inverter [4]

Radiation tolerance to total ionizing dose and single event effects is achieved through layout. A radiation tolerant inverter is shown in Figure 2-15. Total ionizing dose effects are minimized by the use of annular geometry nMOS transistors. This geometry minimizes the shift in V_t by preventing the buildup of trapped charge near the

active region. The transistors are surrounded with highly doped guard rings, which prevent leakage through the field oxide separating the transistors and reduce SEL. High drive strength transistors reduce SEU and lengthens the life cycle at the cost of some efficiency [4]. A benefit of using the HP 0.5 μm process is that it demonstrates a higher tolerance to total ionizing dose than other similar processes. Due to the smaller feature size, its SEU tolerance is lower than larger technologies [25].

2.5 *FFT Comparison*

There are numerous FFT solutions from single board computers to application specific integrated circuits. However, there is no single chip that has all the qualities discussed in this chapter, except for the previous thesis effort. This section presents data found on FFT processors designed for space, low power, speed and a short summary of the previous thesis effort.

2.5.1 *General Purpose FFT Processors*

The most common method of computing an FFT in space is through a general purpose radiation hardened microprocessor. One such processor is the RAD 6000, which is the IBM R/S 6000 processor fabricated by Lockheed Martin's radiation hardened fabrication process [26].

General-purpose machines are a poor example for comparison of power versus speed, as they are generally not optimized for FFT calculations and use a significant amount of power. For example, the RAD 6000 in a low power configuration consumes 2.5 Watts when running at 2.5 MHz, which translates to about 2 MIPS [26].

A radiation hardened processor optimized for Digital Signal Processing (DSP) is a better comparison. Texas Instruments has fabricated a radiation-hardened version of their C40 DSP chip. Specifications on the commercial version are published. The assumption should be made that the space bound version has a lower performance than the commercial version [27].

2.5.2 Low-Power FFT Processors

An outstanding research effort in low power FFT processors is ongoing at Stanford University. An FFT-1024 processor, called Spiffee, was designed and fabricated with low power in mind [28]. It has a high efficiency of 24.7 nJ/Unit Transform. The design is based on a clocked radix-2 decimation in time form of the FFT with a core butterfly processor and uses low V_t transistors. The core of the chip operates at a lower voltage than the I/O circuitry. This design is very sensitive to radiation and noise and is not suitable for space application.

2.5.3 High-Performance Processors

A processor designed for speed alone, called COBRA, was investigated to get an idea what the high-end computational speed is for computing a 1024-point FFT [29]. The COBRA architecture implements a clocked radix-4 decimation in time form of the FFT. It uses multiple butterfly processors connected through a switch matrix. A single COBRA chip can only execute an FFT-64. Sixteen chips are used in parallel to compute an FFT-1024. This design is also not suitable for space, due to the custom multiple chip implementation of the FFT-1024.

2.5.4 Previous Thesis Research

The previous thesis research developed the first implementation of the FASST architecture [2]. The design style used was different than the one that is presented in this thesis. An effort was made to produce a small implementation, which used core ALUs connected to memory banks and coordinated with one-hot style controllers [3]. A 6-bit FFT-4 test chip was fabricated, however the chip unfortunately had design errors which prohibited the determination of any performance data. Simulation data was available detailing the performance of the FFT-16 and extrapolations to an FFT-1024.

2.5.5 Performance Comparison

Table 2-2 gives a comparison of the FFT processors discussed in the previous sections. This data was obtained from an FFT comparison Internet site maintained by Stanford University [30]. Background papers were checked to verify the published data. The data necessary to verify the Spiffee throughput time and the COBRA efficiency was not available. These two results should be considered unreliable estimates [31].

Table 2-2. A Comparison of FFT Processors

Processor Name	Design Feature	Dataword Size & Type	Supply Voltage (V)	FFT-1024 throughput time (μ s)	Efficiency (nJ/Unit Transform)
C40	Space	32-Bit Floating Point	5.0	1298	5704
Spiffee	Low Power	20-bit Fixed Point	3.3	30	24.7
COBRA	Speed	23-bit Fixed Point	5.0	9.5	71.4
FASST	Asynchronous, space & low power	16-bit Fixed Point	3.3	10	120

2.6 *Conclusion*

The goal of this thesis is to design a fast, efficient FFT processor suitable for space. This chapter covered the basics of asynchronous design, FFT theory and radiation hardening of electronics to support the goal of the thesis. The final section compared FFT processors that are the best in their category with the results from a previous thesis. The results of the previous thesis demonstrated that the efficiency and performance achieved using the FASST architecture is exactly what was desired. The estimated FFT-1024 performance and efficiency shows an improvement of two orders of magnitude. This comparison motivates the development of an improved functional FFT using the FASST architecture.

3. Design Overview

The goal of this design is to implement a functional FASST design in silicon. A 16-point FFT was chosen as the base-case to prove that the FASST architecture works and to measure the performance of this implementation. It also utilizes all of the necessary components which are used in arbitrarily large FFTs.

This chapter discusses what the design constraints are and how they impacted the VLSI design. The functionality of the top-level design is described, as well as the functionality of the major components. Chapter Four revisits these components in greater detail down to the gate level.

3.1 Design Constraints

This section covers design constraints imposed on the selection of the cell library, data type and point size. The thesis sponsor recommended these constraints.

3.1.1 Cell Library

The use of a radiation tolerant VLSI cell library was the method chosen to meet the space application requirement. The radiation tolerant library cells provided by AFRL were designed for fabrication on the commercial HP 0.5 μm foundry process line [4].

The size of the radiation tolerant cells is a notable feature. Consider the example of the difference in size between a radiation tolerant minimum-sized inverter and a standard minimum-sized inverter in the Lager IV distribution [13]. A comparison is

illustrated in Figure 3-1. The Lager inverter measures $68\lambda \times 16\lambda$ while the MRC cell measures $124\lambda \times 40\lambda$ [4].

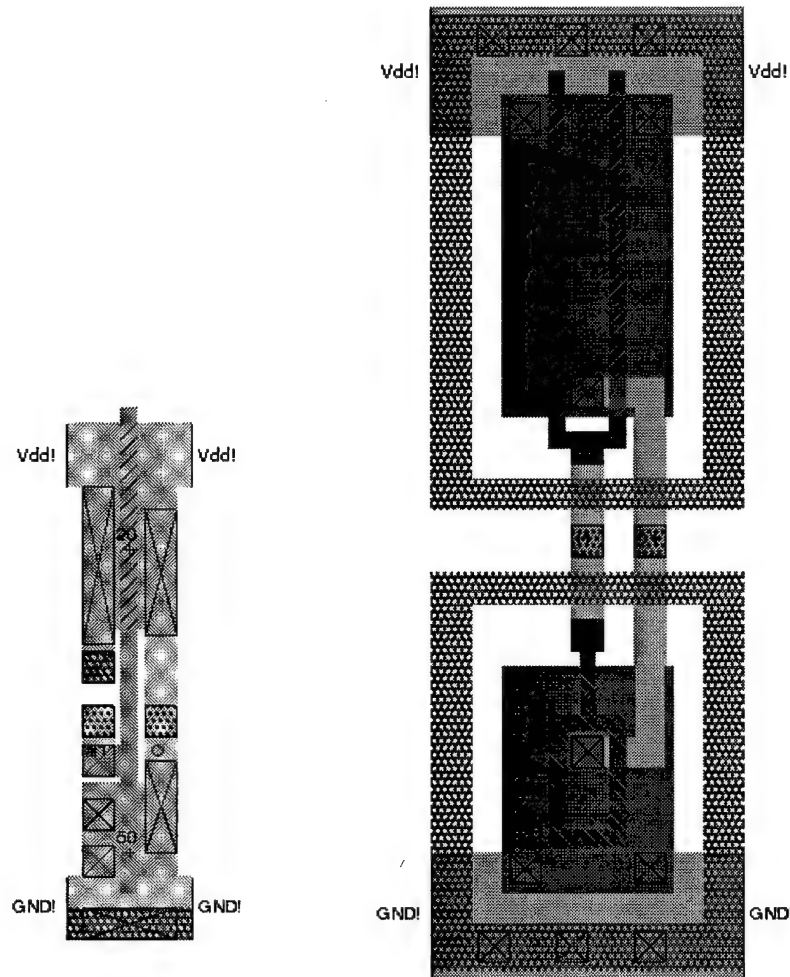


Figure 3-1. Inverter Size Comparison [13][4]

Several design features were engineered into the radiation tolerant library to enable them to be densely packed. The cells are symmetrical and the rings around the cells are allowed to overlap the rings of the nearest neighbors. All of the routing within the cells is at the lowest level metal, which allows routing of multiple traces of higher

level metals directly over the cells. These design features allow the overall design size to be roughly twice as large as a minimum sized design, provided that a high performance router is used [4].

The cells are not optimally power efficient because they are not minimum sized. The choice of using the asynchronous design approach was not only to achieve a faster design, but also to compensate for the additional power consumption of the radiation tolerant library. The HSPICE [32] plot of power versus time in Figure 3-2 indicates that a single radiation tolerant inverter (dashed line) uses more power (area under the curve) than a minimum sized inverter (solid line) by a factor of two. This additional power consumption is necessary to overcome the SEU effects in memory structures discussed in Chapter Two.

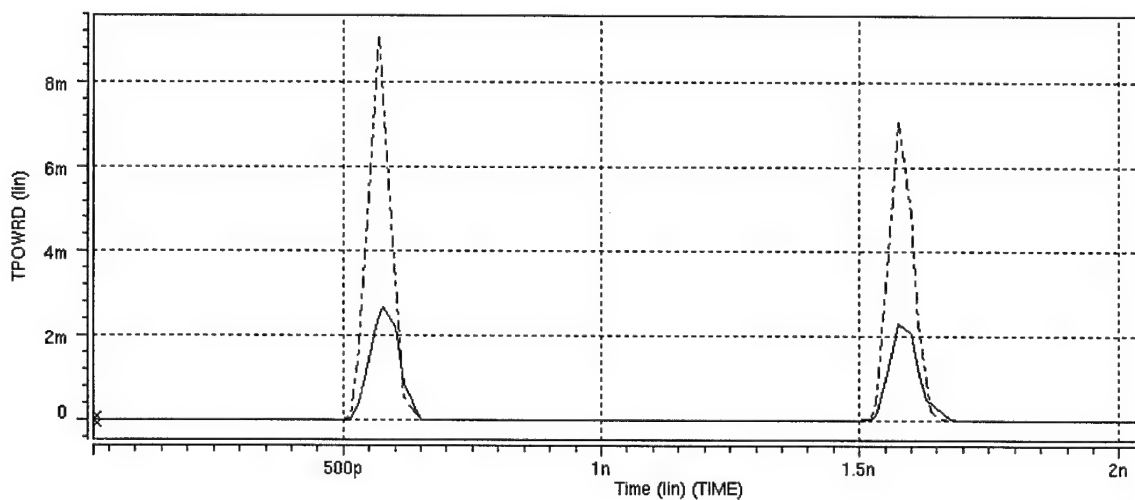


Figure 3-2. Inverter Power Use Comparison

3.1.2 Data Type and Size

The data types used in a DSP processor are integer, fixed point, block-float or floating-point. The size and type of the data greatly affects the design size and

complexity. Typically, a commercial processor will use a floating-point or block-float format to enable a wide product application. If the input and output data are properly scaled, floating point format is not necessary and more efficient designs can be realized. A 16-bit word size for the real and imaginary components was chosen. This data format is shown in Figure 3-3.

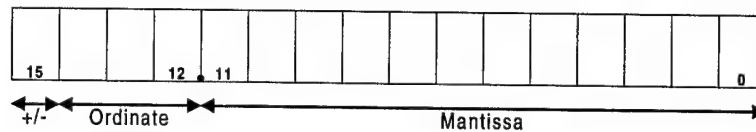


Figure 3-3. 16-bit Data Word Format

This data format chosen allows twelve bits of resolution in the mantissa. This implies that the data word can represent a value of 1000.000000000000_2 (-8.0_{10}) to 0111.111111111111_2 (7.999755859375_{10}).

3.1.3 Project Point Size

A 16-point FFT (FFT-16) is the minimum size used to demonstrate the FASST architecture. The basic building blocks for this base-case are the 4-point FFT (FFT-4) and the complex multiplier [20]. The FFT-4 proved to be an appropriate building block for proving the concept of this thesis design and was fabricated. The FFT-16 may be fabricated in the future by AFRL depending on the success of the FFT-4 fabricated chip.

3.2 FFT-16 Design

The FASST architecture can be applied to any point size, provided that $N=N_1N_2$. To demonstrate the functionality of the FASST architecture, an FFT-16 ($N=16$)

implementation with $N_1 = N_2 = 4$ was chosen. This implies that the FFT-4 will comprise the basic building block of the design.

The $W_N^{m_1 n_2}$ matrix must be used to determine the constants used in the FFT-16. As stated previously, for an FFT-16, $N=16$ and m_1, n_2 will be 0,1,2,3. The $W_{16}^{m_1 n_2}$ matrix is shown in Equation 3-1.

$$W_{16}^{m_1 n_2} = \begin{bmatrix} W_{16}^0 & W_{16}^0 & W_{16}^0 & W_{16}^0 \\ W_{16}^0 & W_{16}^1 & W_{16}^2 & W_{16}^3 \\ W_{16}^0 & W_{16}^2 & W_{16}^4 & W_{16}^6 \\ W_{16}^0 & W_{16}^3 & W_{16}^6 & W_{16}^9 \end{bmatrix} \quad (3-1)$$

Using the $N=16$ vector constant map in Figure 3-4, $W_{16}^{m_1 n_2}$ can be represented with complex constants as shown in Equation 3-2.

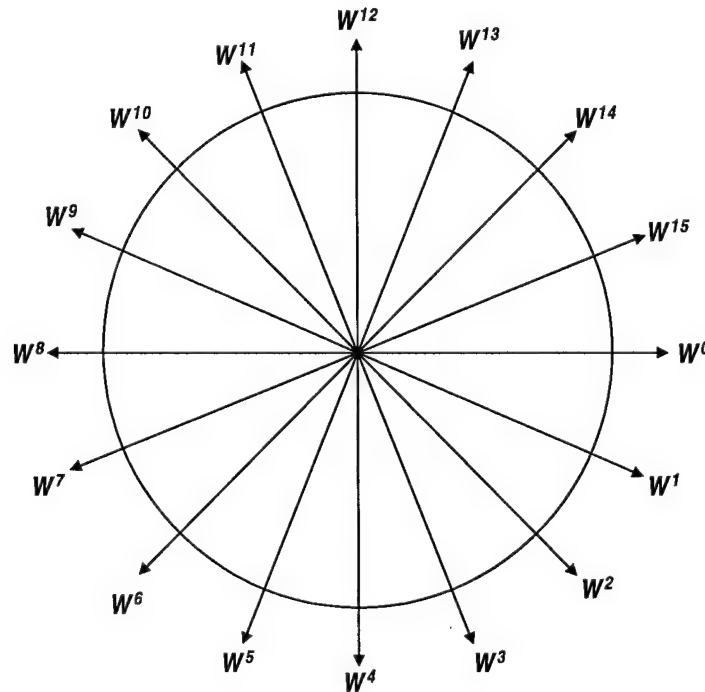


Figure 3-4. FFT-16 Vector Constant Map

$$W_{16}^{m_1 n_2} = \begin{bmatrix} \cos(0) + j \sin(0) & \cos(0) + j \sin(0) & \cos(0) + j \sin(0) & \cos(0) + j \sin(0) \\ \cos(0) + j \sin(0) & \cos\left(\frac{\pi}{8}\right) + j \sin\left(\frac{\pi}{8}\right) & \cos\left(\frac{\pi}{4}\right) + j \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{3\pi}{8}\right) + j \sin\left(\frac{3\pi}{8}\right) \\ \cos(0) + j \sin(0) & \cos\left(\frac{\pi}{4}\right) + j \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{2}\right) + j \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{3\pi}{4}\right) + j \sin\left(\frac{3\pi}{4}\right) \\ \cos(0) + j \sin(0) & \cos\left(\frac{3\pi}{8}\right) + j \sin\left(\frac{3\pi}{8}\right) & \cos\left(\frac{3\pi}{4}\right) + j \sin\left(\frac{3\pi}{4}\right) & \cos\left(\frac{9\pi}{8}\right) + j \sin\left(\frac{9\pi}{8}\right) \end{bmatrix} \quad (3-2)$$

Now that all the components are derived and defined, the overall generic block diagram in Figure 2-12 applied to the FFT-16 (with $N_1 = N_2 = 4$) results in a data flow block diagram (with simplified constants) as shown in Figure 3-5. It is interesting to note that other texts have presented this data flow block diagram with no implication made that can be implemented in an asynchronous nature, as in the FASST architecture [33].

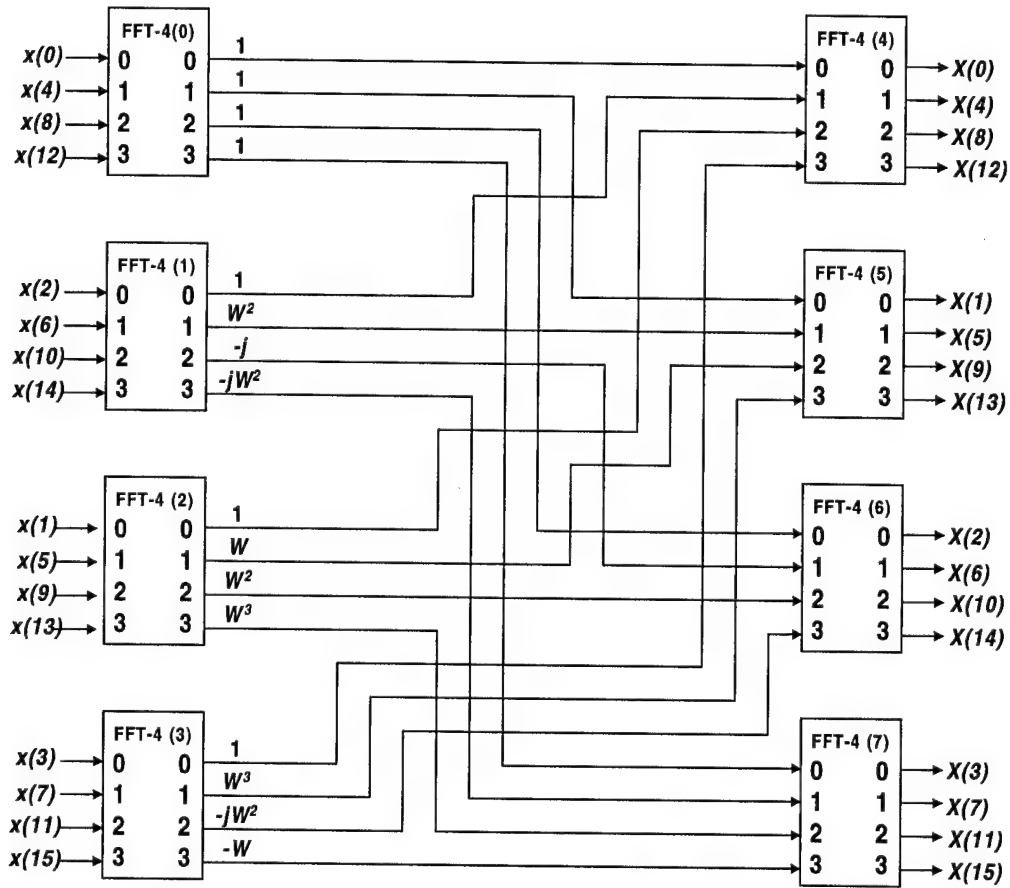


Figure 3-5. FFT-16 Data Flow Block Diagram

The data flow block diagram in Figure 3-5 implies that an FFT-16 will require eight FFT-4 components (0 through 7) and only three complex multipliers because the outputs of FFT-4 (0) are multiplied by one.

Additional components are needed to route the control signals and store the constant values. The decimator and expander are responsible for routing incoming data and ordering outgoing data, respectively. The crossbar is responsible for handling the interconnection between the input and output FFT-4 units.

Sections 3.2.1 through 3.2.6 give an overview of the composition and function of each major component of the FFT-16. Section 3.2.7 combines the components and describes the FFT-16 operation at the top level.

3.2.1 FFT-4

The FFT-4 computes the four-point FFT of the input sequence. The derivation of this base-case FFT is presented to illustrate how the math correlates to a physical design.

Using Equation 3-3, the FFT-4 is described as in Equation 3-3.

$$X(m) = W_4 x(n) \quad (3-3)$$

Equation 3-1 can be expanded into matrix format to show the values of the W_4 coefficients. This expansion is shown in Equation 3-4.

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^1 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (3-4)$$

By using the W_4 vector map in Figure 3-6, the W_4 constants can be evaluated (i.e. $W^0 = 1$, $W^1 = -j$, etc. where $j = \sqrt{-1}$). The vector map is periodic, which means, for example, $W^0 = W^4$. The new expression is shown in Equation 3-5 with W values replaced with constants.

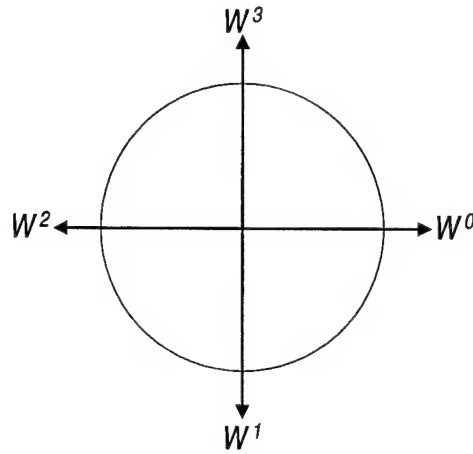


Figure 3-6. FFT-4 Vector Constant Map

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (3-5)$$

Separating the matrix into individual equations yields Equations 3-6 through 3-9.

$$X(0) = x(0) + x(1) + x(2) + x(3) \quad (3-6)$$

$$X(1) = x(0) - jx(1) - x(2) + jx(3) \quad (3-7)$$

$$X(2) = x(0) - x(1) + x(2) - x(3) \quad (3-8)$$

$$X(3) = x(0) + jx(1) - x(2) - jx(3) \quad (3-9)$$

Inspection of these equations reveals that there are twelve complex add operations and four complex multiplies. The complex multiplies are eliminated by substitution giving 16 additions or subtractions. By letting $a = x(0) + x(2)$, $b = x(1) + x(3)$, $c = x(0) - x(2)$,

and $d = x(1) - x(3)$, Equations 3-6 through 3-9 are represented by Equations 3-10 through 3-13.

$$X(0) = a + b \quad (3-10)$$

$$X(1) = c - jd \quad (3-11)$$

$$X(2) = a - b \quad (3-12)$$

$$X(3) = c + jd \quad (3-13)$$

Expressing the complex variables as real and imaginary components yields Equations 3-14 through 3-17.

$$\text{Re}\{X(0)\} + j\text{Im}\{X(0)\} = \text{Re}\{a\} + j\text{Im}\{a\} + \text{Re}\{b\} + j\text{Im}\{b\} \quad (3-14)$$

$$\text{Re}\{X(1)\} + j\text{Im}\{X(1)\} = \text{Re}\{c\} + j\text{Im}\{c\} - j(\text{Re}\{d\} + j\text{Im}\{d\}) \quad (3-15)$$

$$\text{Re}\{X(2)\} + j\text{Im}\{X(2)\} = \text{Re}\{a\} + j\text{Im}\{a\} - (\text{Re}\{b\} + j\text{Im}\{b\}) \quad (3-16)$$

$$\text{Re}\{X(3)\} + j\text{Im}\{X(3)\} = \text{Re}\{c\} + j\text{Im}\{c\} + j(\text{Re}\{d\} + j\text{Im}\{d\}) \quad (3-17)$$

Finally, expressing $\text{Re}\{X\}$ and $\text{Im}\{X\}$ separately with j factored through, the final Equations 3-18 through 3-25 are realized.

$$\text{Re}\{X(0)\} = \text{Re}\{a\} + \text{Re}\{b\} \quad (3-18)$$

$$\text{Im}\{X(0)\} = \text{Im}\{a\} + \text{Im}\{b\} \quad (3-19)$$

$$\text{Re}\{X(1)\} = \text{Re}\{c\} + \text{Im}\{d\} \quad (3-20)$$

$$\text{Im}\{X(1)\} = \text{Im}\{c\} - \text{Re}\{d\} \quad (3-21)$$

$$\text{Re}\{X(2)\} = \text{Re}\{a\} - \text{Re}\{b\} \quad (3-22)$$

$$\text{Im}\{X(2)\} = \text{Im}\{a\} - \text{Im}\{b\} \quad (3-23)$$

$$\text{Re}\{X(3)\} = \text{Re}\{c\} - \text{Im}\{d\} \quad (3-24)$$

$$\text{Im}\{X(3)\} = \text{Im}\{c\} + \text{Re}\{d\} \quad (3-25)$$

The FFT-4 can then be accomplished with only 16 individual add or subtract operations and no complex multiplies. No complex multiplies makes the FFT-4 an ideal base-case. Figure 3-7 illustrates the final layout of addition and subtraction operations.

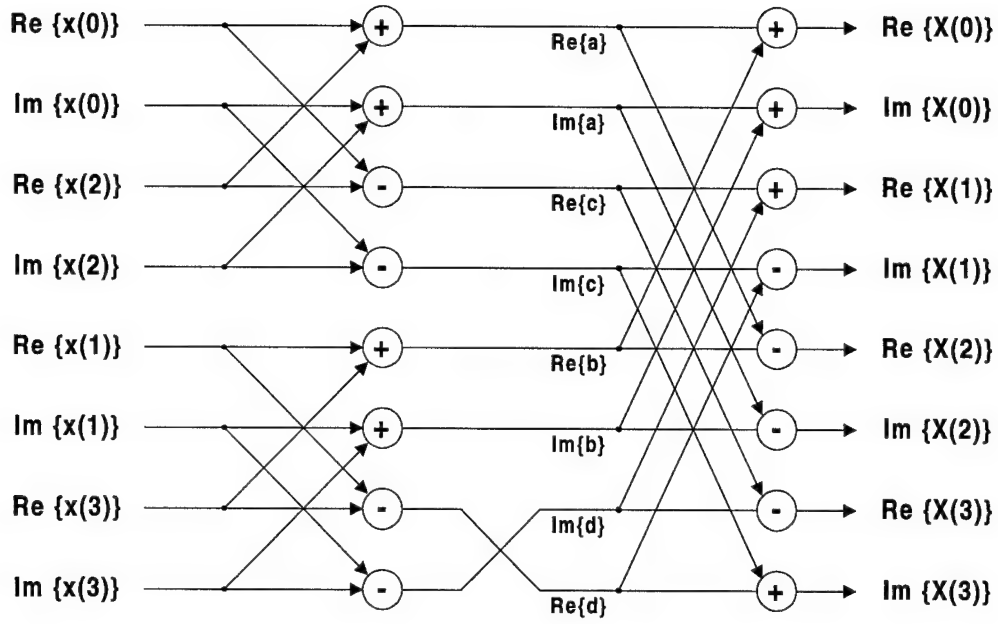


Figure 3-7. FFT-4 Block Diagram For 16 Add/Subtract Operations

3.2.2 Complex Multiplier

The hardware implementation of a complex multiply must model Equation 3-26, which is the mathematical definition of the multiplication of two complex values X and Y . Equation 3-27 gives the equivalent method for multiplication in hardware that has separate real and imaginary data buses, as in this design.

$$XY = (\text{Re}\{X\} + j\text{Im}\{X\})(\text{Re}\{Y\} + j\text{Im}\{Y\}) \quad (3-26)$$

$$\begin{aligned} XY &= (\text{Re}\{X\} + j\text{Im}\{X\})(\text{Re}\{Y\} + j\text{Im}\{Y\}) \\ &= \text{Re}\{X\}\text{Re}\{Y\} - \text{Im}\{X\}\text{Im}\{Y\} + j(\text{Re}\{X\}\text{Im}\{Y\} + \text{Re}\{Y\}\text{Im}\{X\}) \end{aligned} \quad (3-27)$$

The design pursued in this research uses asynchronous building blocks to produce a fully asynchronous complex multiplier. The complex multiplier block diagram is shown in Figure 3-8.

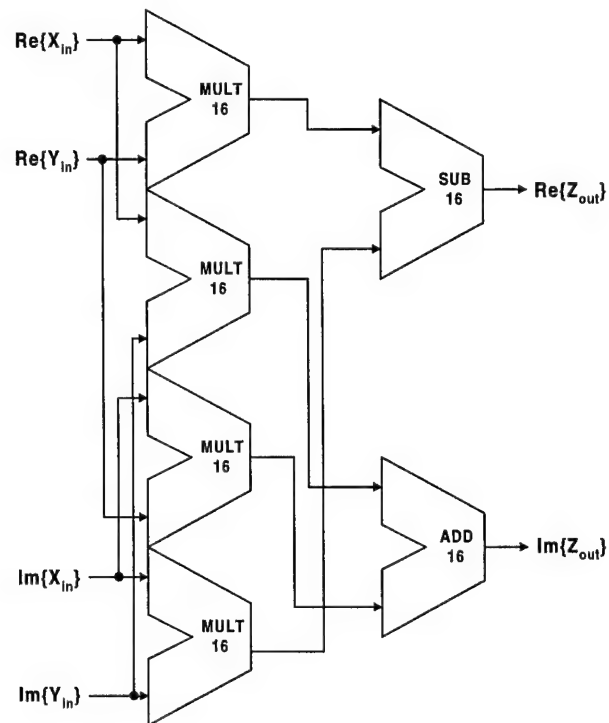


Figure 3-8. Complex Multiplier Layout

3.2.3 Decimator

The decimator is a functional block that takes the ordered input data ($x(0)$ through $x(15)$) and routes them to the respective FFT-4 blocks shown in Figure 3-5. The values $x(0)$, $x(4)$, $x(8)$ and $x(12)$ are sent to the first FFT-4. The values $x(1)$, $x(5)$, $x(9)$ and $x(13)$ are sent to the second FFT-4. The next two groups of values are sent to the third and fourth FFT-4 blocks, respectively.

This block is implemented with a two-bit Johnson counter that routes the FFT-16 REQIN and ACKIN signals to the REQIN and ACKIN lines of the first four FFT-4s. The connections are shown in Figure 3-9. The data is transferred from the input bus to the FFT-4s through a shared input bus corrected for fanout. This design would require modification if it were used in a high-speed design.

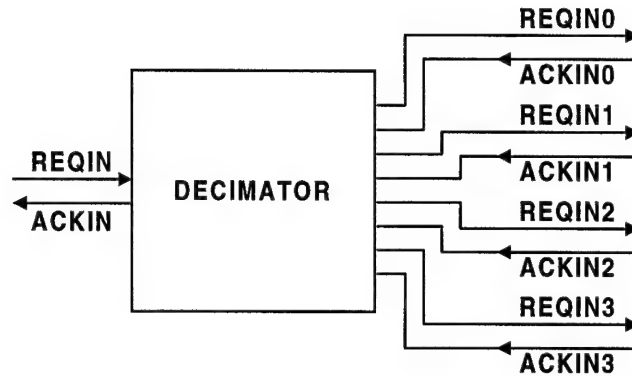


Figure 3-9. Decimator Control Signals

3.2.4 Expander

The expander performs the exact opposite operation of the decimator described in the previous section. It takes the output stream from the final set of FFT-4 blocks and orders the values so they appear as $X(0)$ through $X(15)$ on the output. When $X(0)$ becomes available from FFT-4 (4), it is sent out first. When $X(1)$ becomes available from FFT-4 (5), it is sent out next. The sequence is repeated until all 16 values are sent out.

Components similar to the ones used in the decimator are used in the expander to route the control signals. The control signals are shown in Figure 3-10. The data is routed through a mux controlled by the expander hardware.

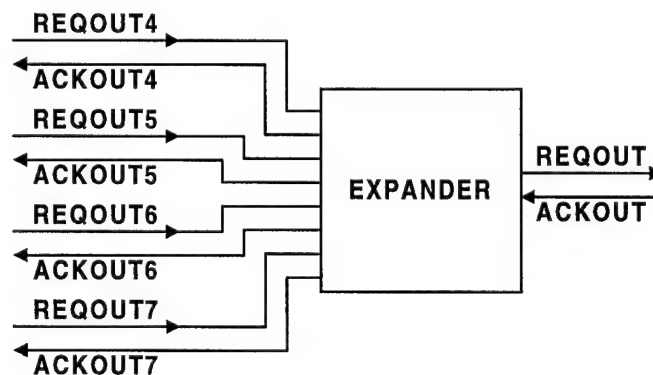


Figure 3-10. Expander Control Signals

3.2.5 "Crossbar"

The "crossbar" is responsible for handling the interconnections between the input FFT-4s and the output FFT-4s, as shown in Figure 3-5. Output values from the input FFT-4s are handled using the same method as the expander. In effect, the four pairs of REQOUT/ACKOUT signals are reduced to a single pair of REQOUT/ACKOUT signals. Then, according to Figure 3-5, the first four available values are sent to FFT-4 (4). The next four are sent to FFT-4 (5), and so on.

The crossbar is implemented with an expander coupled to a divide by four element. The data is passed on a shared bus with tristate buffers that the crossbar controls. The signals are shown in Figure 3-11.

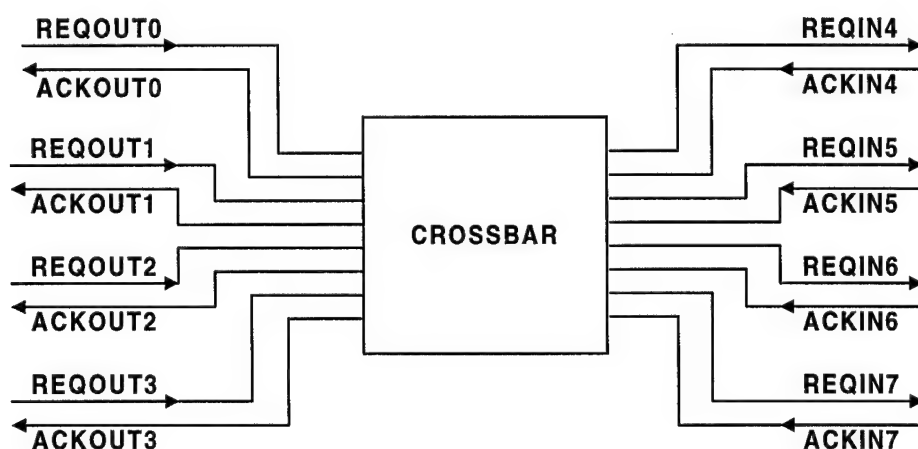


Figure 3-11. Crossbar Control Signals

3.2.6 Constant Banks

There are three constant banks, which store the values of the complex constants for multiplication with the first stage FFT-4 results, as shown in Figure 3-5. These constant banks are created with combinational logic controlled by an AFSM.

3.2.7 Putting the FFT-16 Together

Now that each component has been described, the top-level picture can be presented. The components interconnected by the control and data paths are shown in Figure 3-12.

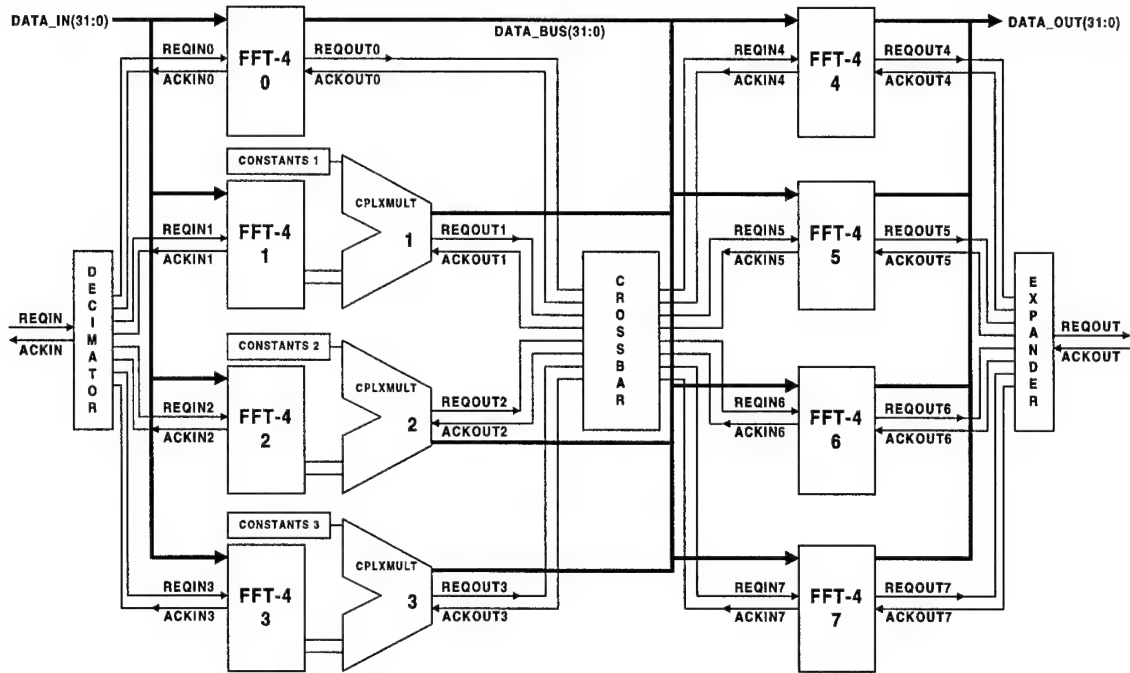


Figure 3-12. FFT-16 Components

3.3 Design Conclusion

This chapter outlined the hardware required for an FFT-16 using the FASST architecture. The FASST architecture produces a layout that is highly localized and reuses major components, as seen in Figure 3-12. This lays the groundwork for the detailed discussion of each component in Chapter Four.

4. Design Implementation

This chapter presents the design details of each component discussed in Chapter Three. The chapter begins by looking at the FFT-16 top level design and then discusses the top-level operation. Other designs considered for each component are also presented.

4.1 FFT-16

The FFT-16 executes the FFT on an input data stream. The top-level block diagram is shown in Figure 4-1. The complex values $x(0)$ through $x(15)$, are fed in order on the DATA_INR (real component) and DATA_INI (complex component) input buses. The transformed values $X(0)$ through $X(15)$ appear on the output buses DATA_OUTR and DATA_OUTI in order as evaluated by Equation 2-8 for $N=16$ and $N_1=N_2=4$. Each data input and output cycle takes place with a four-cycle handshake.

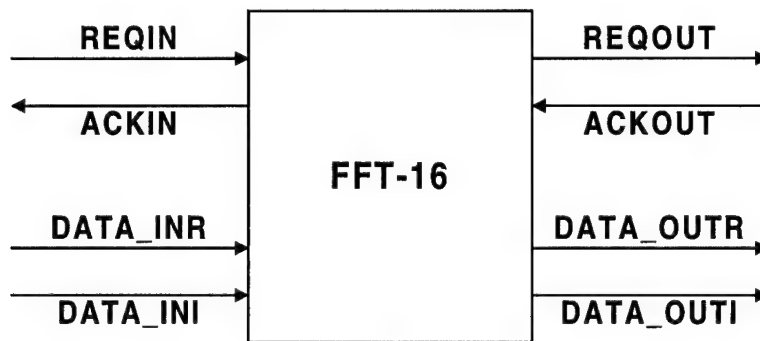


Figure 4-1. FFT-16 Top Level

The FFT-16 is composed of six major components, as shown in Figure 4-2. The FFT-4, complex multiplier, decimator, expander and crossbar are discussed in detail in the following sections.

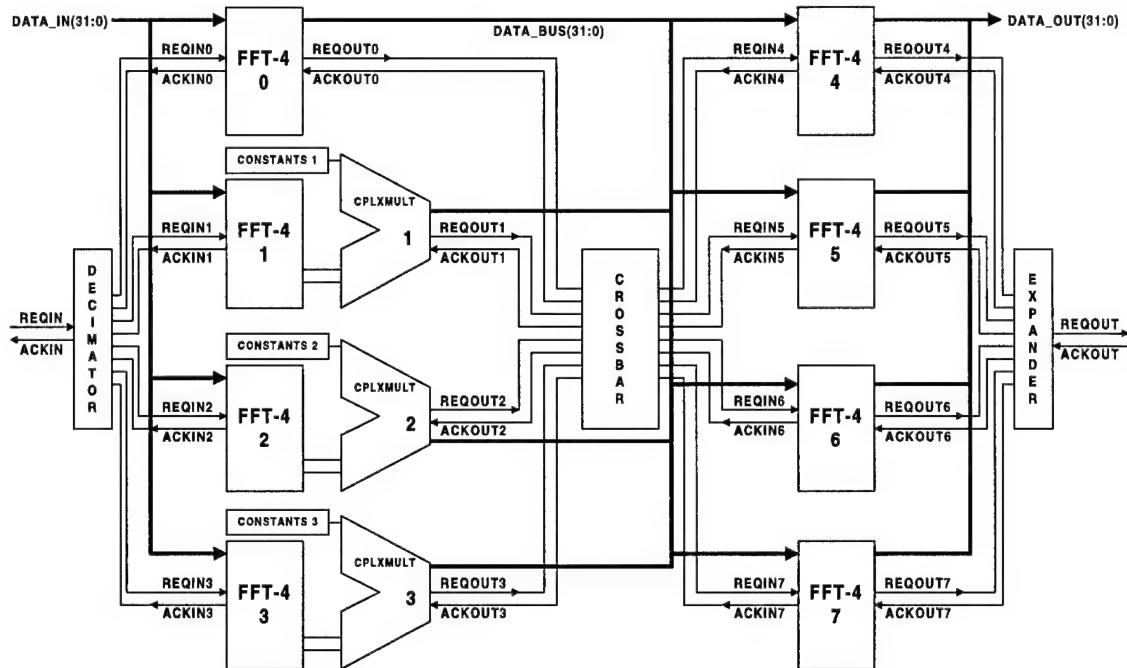


Figure 4-2. FFT-16 Components

4.2 FFT-4

Previous research efforts have explored several architectures of the FFT-4. The most recent effort used 16 registers and 2 ALUs (add/subtract units) [2]. The author's recommendations for continuation suggested that there might be more simple approach to the one that was implemented. The design suggested was to use 16 dedicated add and subtract units to accomplish the 16 add and subtract operations for the FFT-4 instead of two ALUs.

The idea of simplification was taken a step further in this research. A design was pursued that used only 8 latches to latch the inputs and 8 ALUs. The reduction from 16 to 8 adders was possible by realizing that Equations 3-16 through 3-23 can be reduced to four equations by simply switching the operator in the matching equations. For example, Equation 3-16 is identical to 3-20 with the exception of the operator. The control for the design proved to be more complex than the 16 element option, but it produced a design that had several advantages. The area was reduced by over 50% and the large fan-outs present were also reduced by 50%. The energy efficiency does not change much as the same number of switching operations takes place although a small gain is realized due to the decreased circuit fan-out. Figure 4-3 illustrates the interconnection of the eight ALUs used to accomplish the FFT-4 calculation.

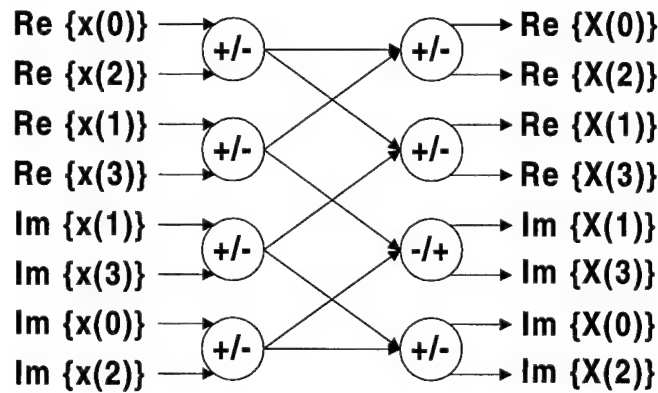


Figure 4-3. FFT-4 Block Diagram Using 8 ALUs

This design is much smaller in area than the one previously reported [2]. Table 4-1 highlights the difference in area between the previous design and the one developed in this research effort. The area of the 2-ALU/16-Register design is extrapolated from the original size of 6 bits to 16 for comparison.

Table 4-1. FFT-4 Area Comparison

Design	Dimensions	Total Size
2-ALUs, 16-Registers	$9242\lambda \times 8925\lambda$	$82.5 \times 10^6 \lambda^2$
8-ALUs, 8-Latches	$6148\lambda \times 5942\lambda$	$36.5 \times 10^6 \lambda^2$

The major components of the FFT-4 are the input latches, ALUs, output multiplexor (mux), and the control units as shown in Figure 4-4. A test mux is included on the fabricated FFT-4 to incorporate design for testability. The test mux is not included in the FFT-4 design used in the FFT-16.

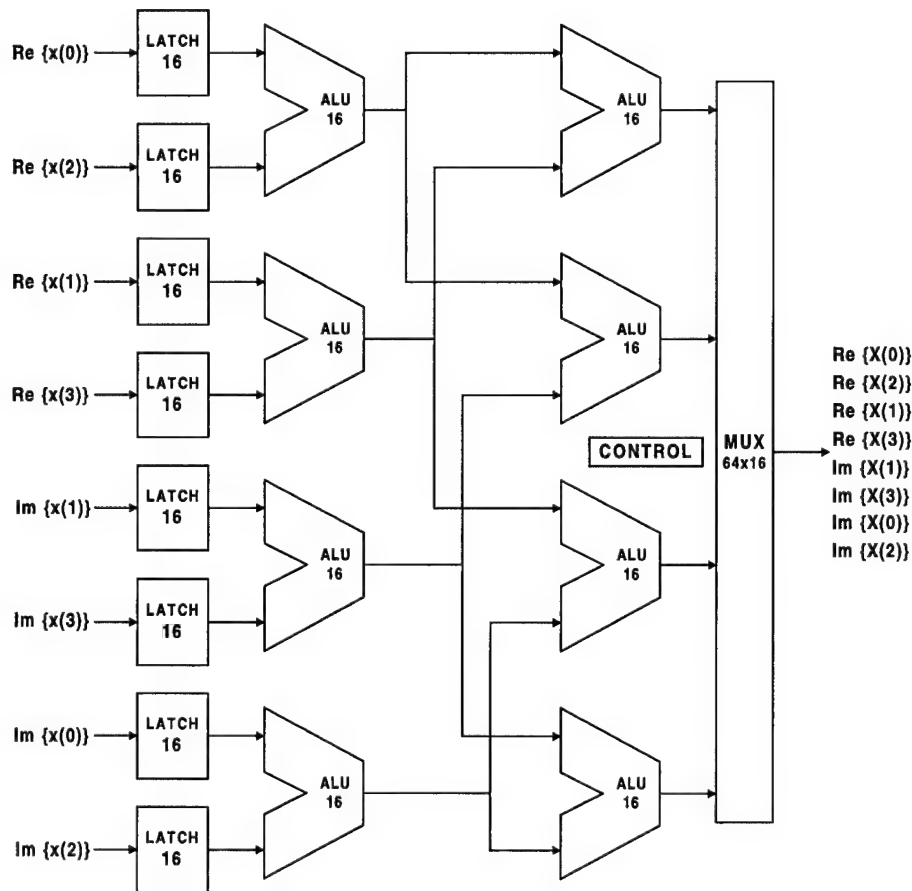


Figure 4-4. FFT-4 Initial Design

The input latches are the only memory structures in the FFT-4. They latch the input data as it appears on the input bus. The ALUs are 16-bit asynchronous add/subtract units. Instead of a traditional ALU which is clocked at the worst-case rate, it has completion detection circuitry which enables a better than worst case flow of data through each ALU. The output mux is an array of simple 2×1 mux cells that route the results of the four output ALUs to the output bus. The test mux is also an array of muxes that route internal control and data signals to test ports.

4.2.1 Input Latches

The input latches shown in Figure 4-4 are composed of 16 one-bit latches as shown in Figure 4-5.

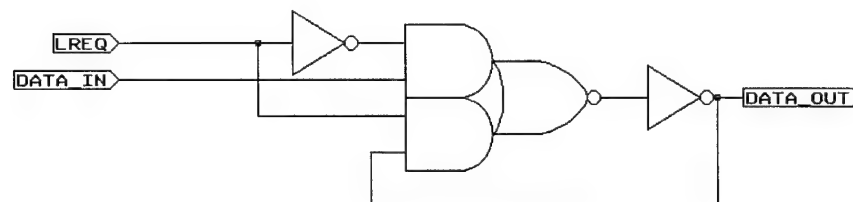


Figure 4-5. One-bit Latch Cell

The delay through the latches is nearly constant. The ideal asynchronous design methodology is the fundamental mode bounded delay model. A simple delay element is used to signal LACK at a safe time after the LREQ signal is asserted. HSPICE was used to determine the correct number of inverters needed to represent the delay necessary for each latch. The HSPICE simulation is shown in Figure 4-6. For this element, the simulation showed that it took a worst-case delay of 1.13 ns to latch the input data. To

model this delay, 14 inverters are needed, as each inverter has a delay of 0.08 ns. Two additional inverters are used to provide a margin of safety.

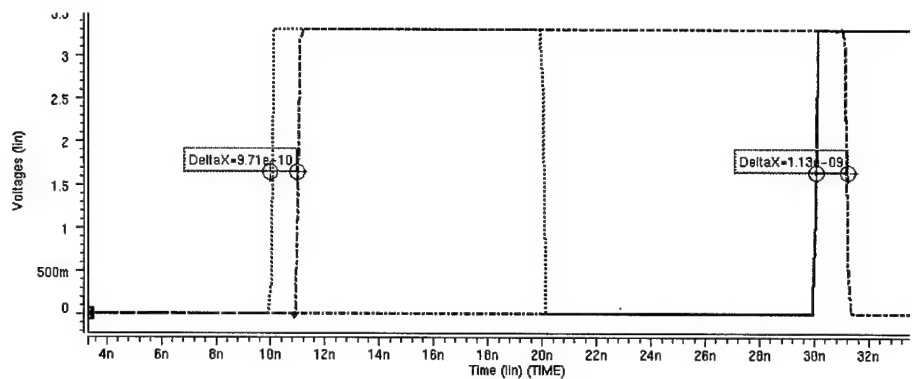


Figure 4-6. HSPICE Simulation of Register Latch Time

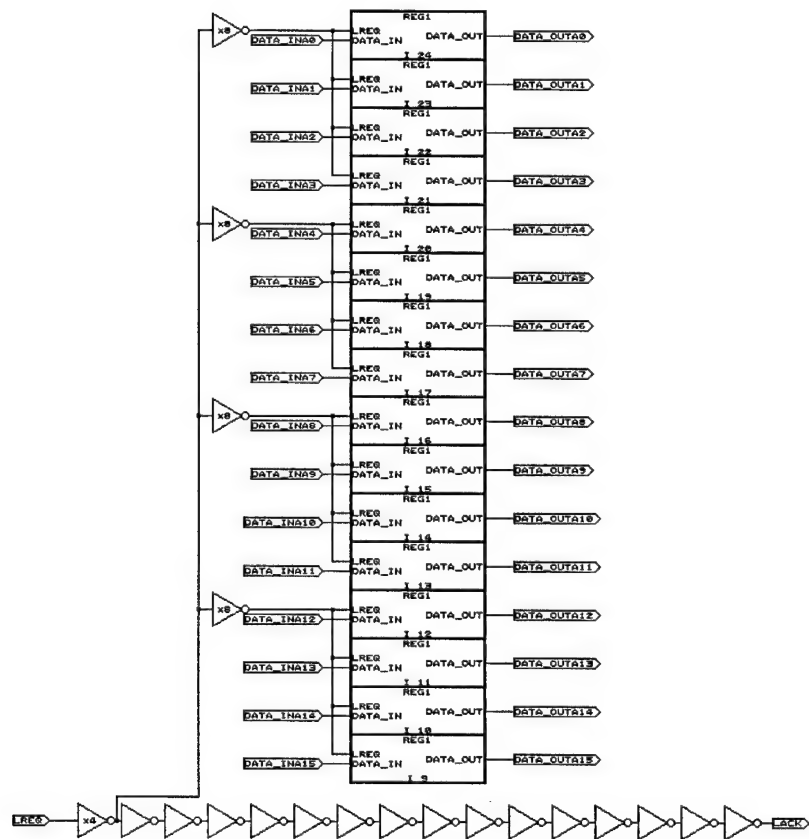


Figure 4-7. LATCH16 Schematic

Only one delay element is necessary and is included in the top level LATCH16 component. Inverters with a fanout of eight were used to correct for fanout of the LREQ signal. The LATCH16 schematic is shown in Figure 4-7.

4.2.2 *Asynchronous ALU*

The asynchronous ripple-carry ALU has an execution time that can vary widely from a minimum to a maximum case. The ripple-carry design was chosen due to its small size and simplicity when compared to other adder schemes. If one assumes that a random distribution of data is processed by the ALU, the average computation time will lie somewhere between the minimum and maximum time. If this were a synchronous circuit, the computation time would be fixed at the worst-case time. Therefore, the delay insensitive design methodology was applied and completion detection circuitry was added.

Implementing completion circuitry adds complexity, size, and increases the required power. The initial 1-bit ALU design used in the previous research was the starting point for this design [2]. It is based on a dual-rail asynchronous adder unit design developed at the University of Manchester [15]. The principle of operation defines that each 1-bit stage (shown in Figure 4-8) of the adder will either have a carry out or no carry out. These signals are designated as COUT and NOCOUT. When either of these lines is raised, the stage can be considered “done.” The ALU reports completion with an ACK signal when all stages are done.

The ALU developed in the previous effort relied on a ripple type reset of the circuit after each calculation to clear out the COUT and NOCOUT values for each stage

[2]. In this design, an alternative was chosen, which resets the COUT and NOCOUT signals with a pair of NOR gates that are tied to ALU16 REQ signal after each calculation to improve the overall execution time of the ALU.

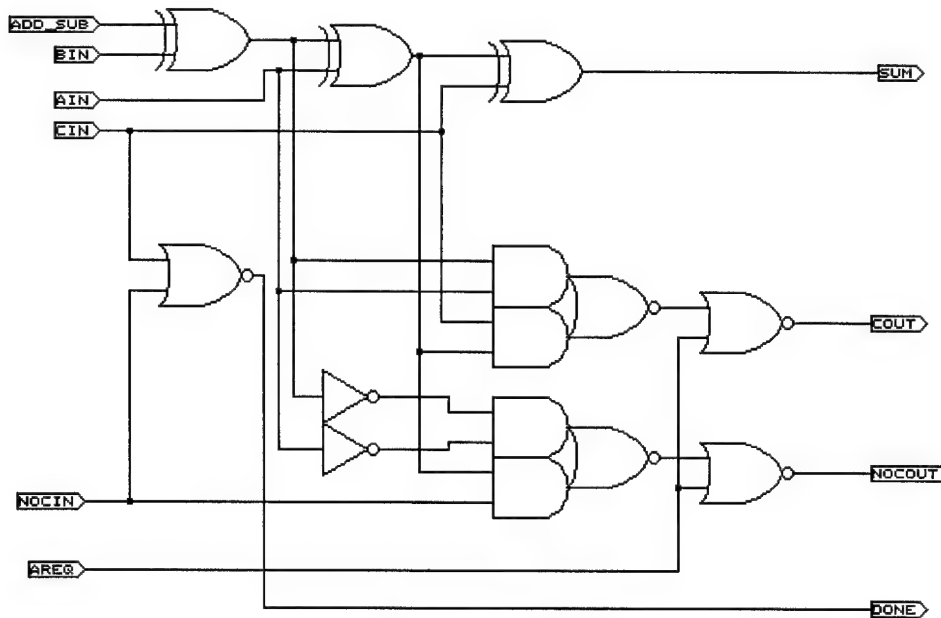


Figure 4-8. Final ALU 1-bit Stage

Three additional components are necessary to complete the design of the ALU. To make the unit capable of subtracting, the exclusive OR function (XOR) is applied to one of the input lines (B was arbitrarily chosen) with the ADD/SUB line being the other input line. Adding the XOR gate simply inverts the B value. Coupled with the component described in the next paragraph, the ALU can execute the subtract function.

An initialization stage was necessary to set the first stage CIN or NOCIN values according to the ADD_SUB and REQ values. The truth table is shown in Table 4-2 and the corresponding circuit is shown in Figure 4-9.

Table 4-2. ALU Initialization Truth Table

REQ	ADD_SUB	COUT	NOCOUT
0	X	0	0
1	0	0	1
1	1	1	0

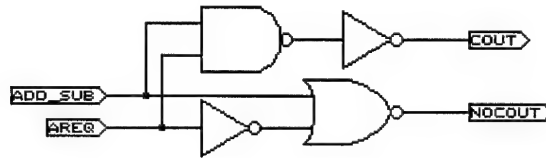


Figure 4-9. ALU Initialization Stage Circuit

The ALU16 consists of one ALU initialization stage along with 16 1-bit ALUs. The NOCOUT and COUT from each stage is connected to the next stage's NOCIN and CIN. A NOR-NAND tree is used to combine the done signals from each stage into the ALU16 ACK signal.

4.2.3 Output Multiplexor

To control which output of the second stage of ALUs is seen on the FFT-4 output, a mux is used. A basic 4x1 mux is constructed from three library cell 2x1 muxes. Sixteen of these muxes are combined to handle all the output values. Figure 4-10 shows the 4x1 mux design.

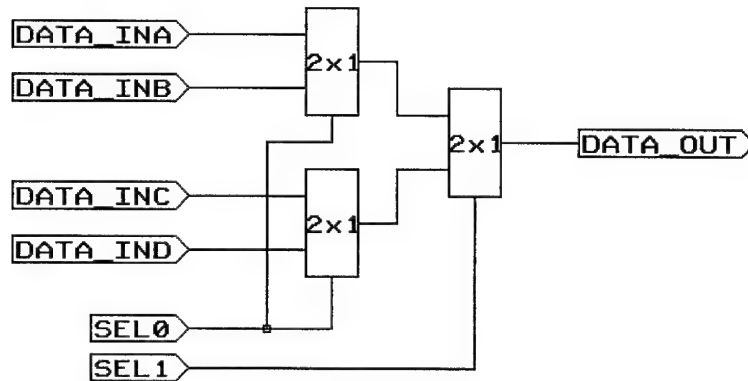


Figure 4-10. 4x1 Mux

Similar to the input latches, a delay element is used to implement the ACK signal for the output multiplexor. This type of delay scheme is appropriate because there is very little variation in completion time when switching between the four mux states. Although a delay element is used in the fabricated FFT-4 design, later investigation found that a delay element on a mux could be eliminated. Changing the states early enough in the control sequence assures that the proper values are seen on the output when the REQOUT signal is given for the FFT-4.

4.2.4 FFT-4 Control Units

Development of the AFSMs necessary for the FFT-4 was an iterative process. Originally, an input and an output controller were designed in behavioral VHDL. They were coupled so that new input data could not overwrite output data not yet sent out. These controllers turned out to be too large to be synthesized by 3D. The controllers were broken up using Shannon decomposition until they could be synthesized and simulated successfully at the structural VHDL level. The final design has nine small controllers, which are all interconnected.

4.2.5 Test Multiplexor

For the fabricated FFT-4 design, a test mux was added to the design to allow probing of internal signals by selecting a bank of test signals and viewing the output through bi-directional pads. Sixteen 4x1 multiplexors were used to allow testing of 64 internal signals.

The selection of the test signals was based on what would be needed to isolate a problem if the test chip failed. All of the AFSM control signals were included to test for any control path problems. Two output data bits and their complete upstream paths were included to test any data path problems. The signals that can be observed are listed in the specification sheet of the FFT-4 chip in Appendix B.

4.2.6 Final FFT-4 Design

It became apparent when assembling the components for the FFT-16 that the design of the FFT-4 had to be changed. Although this resulted in a difference between the fabricated chip design and the final design used in the FFT-16, the test chip still effectively serves the purpose of confirming that the controllers, ALUs and radiation tolerant library function properly.

The new FFT-4 design used in the FFT-16 is the same basic structure that was presented in Figure 4-4. However, several modifications were made for better integration into the FFT-16 design. The control sequence was modified to allow the in order input of data. The output mux was split in half to make two 32x16 muxes, allowing the output of the real and imaginary results at the same time. To handle the changes, four simpler controllers replaced the original nine controllers. The AFSM descriptions of the controllers are given in Appendix C. One of the controllers was reused while three new

controllers were developed. The only drawback to these modifications is that the output order becomes $X(0)$, $X(2)$, $X(1)$, $X(3)$ to achieve minimum switching. Both the crossbar and the reorder register handle this out-of-order sequence.

Another enhancement made was additional fanout corrections throughout the circuit. The final FFT-4 design is almost exactly the same area as the original FFT-4 shown in Figure 4-4. Figure 4-11 shows the layout of the final FFT-4 design.

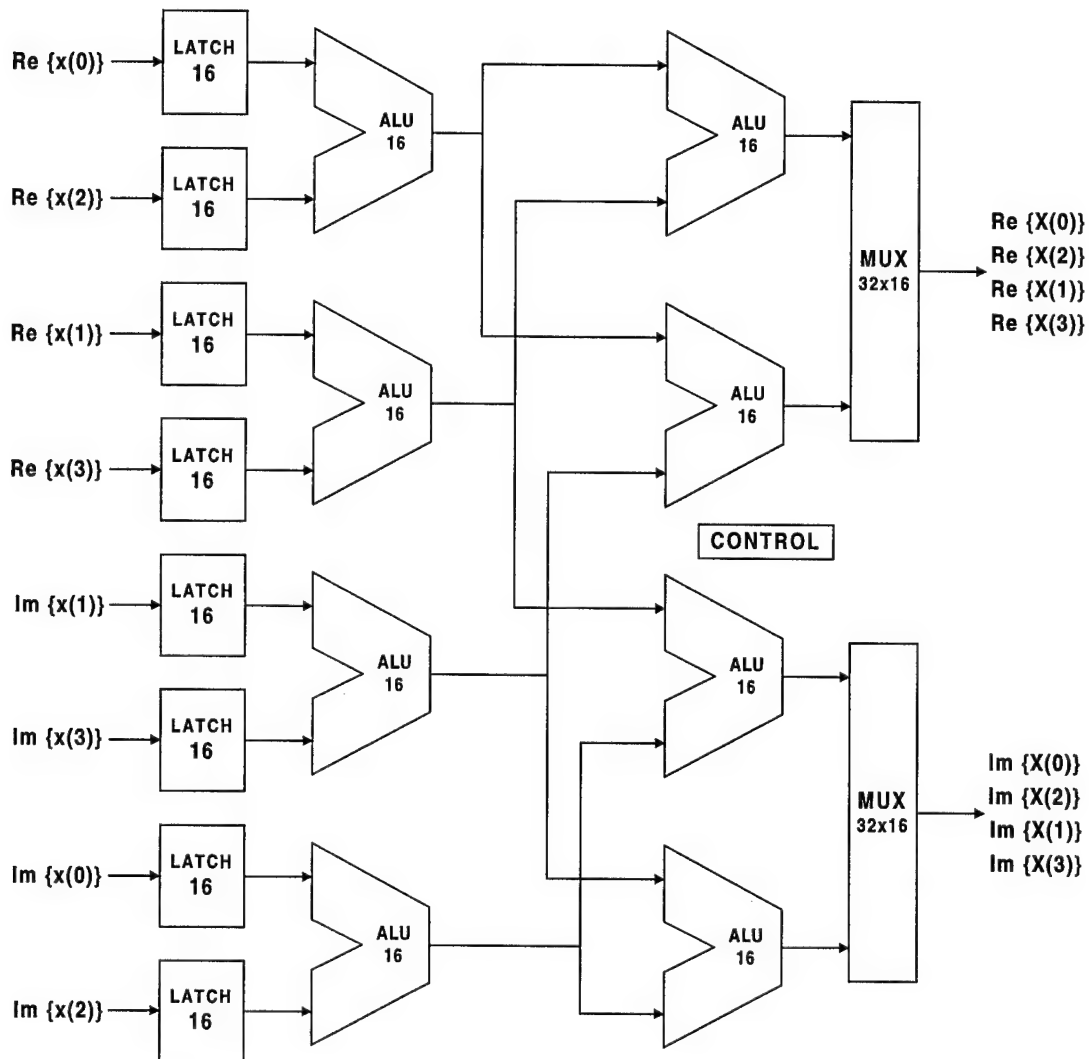


Figure 4-11. Final FFT-4 Design

The overall throughput time of the final FFT-4 design is 75% faster than the first design in this effort due to the availability of the real and complex components at the same time. The area of the final design is slightly smaller due to the reduced number of controllers and muxes.

4.3 *Complex Multiplier*

The design of the complex multiplier used in the previous research was based on a design developed at AFIT [2][34]. The design is based on the Booth multiplication algorithm, which is suitable for asynchronous implementation [35].

The design pursued in this effort employs a radix-4 Booth encoded scheme in the real multipliers and has no component reuse. This design is twice as fast as the radix-2 implementation with a trivial impact to area, due to a slightly more complex controller. The complex multiplier also has a provision for directly forwarding the input to the output for a fast multiply by one, which saves power and gives a higher throughput. This was implemented because one of the four constants in each constant bank (described in Section 4.4) is one. The complex multiplier is composed of four 16-bit real multipliers, one subtract unit, one add unit and a multiply by one mux to accomplish the complex multiply operation in Equation 3-27. Figure 4-12 shows the components of the complex multiplier.

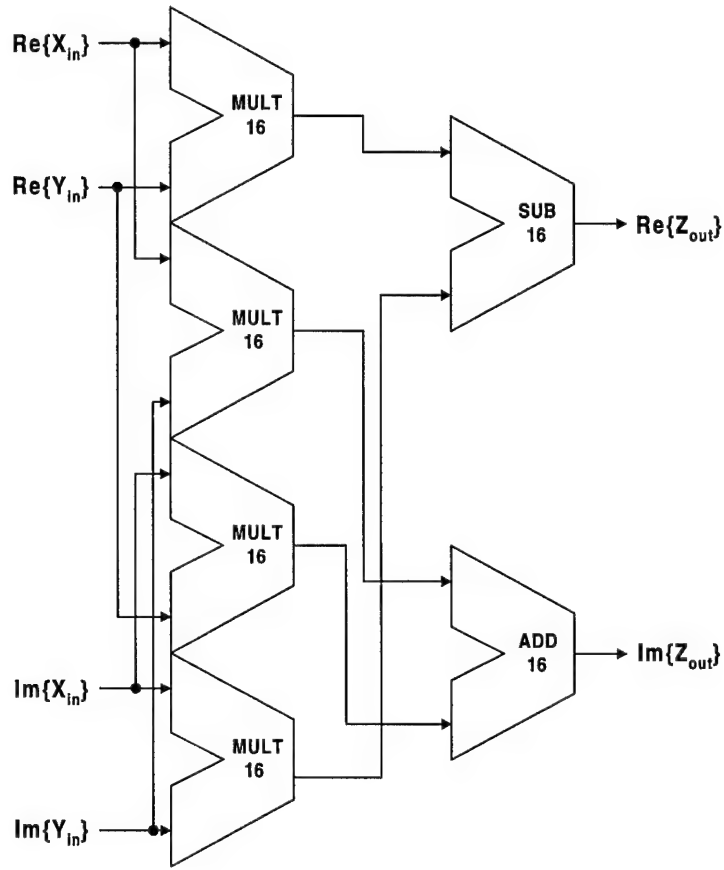


Figure 4-12. Complex Multiplier Block Diagram

The radix-4 Booth encoding was chosen over the baseline radix-2 encoding because it reduces the maximum possible ALU operations from 16 to 8 for 16-bit data. Higher order radix algorithms are possible, but make the design more complex [35].

Multiplying two 16-bit words together produces a 32-bit result. Realizing that this complicates the design by doubling the data width of components downstream from the multiplier, the output of the multiplier was normalized. Normalization is achieved by selecting the 16 output signals that represent the original data format shown in Figure 3-3. Overflow is not a concern, as the inputs are always fractional in an FFT producing a result less than the input magnitude.

The main components of the complex multiplier are the 2's complement radix-4 Booth multiply units, the add unit and the subtract unit. These three components are described in detail in the next sections.

4.3.1 Multiply by One Mux

A set of muxes on the output of the complex multiplier allows the direct forwarding of the X_{in} value to the output under a multiply by one condition. This condition is set by the constant bank, which occurs once every fourth multiply.

4.3.2 Add and Subtract ALUs

Two dedicated ALUs were modified for use with the complex multiplier. They are identical to the ALU16 design in the FFT-4. Because each ALU performs an add or subtract, useless gates were removed from each ALU to quicken its operation and reduce its size.

4.3.3 Radix-4 Booth Encoded Fixed-Point Multiplier

The heart of the complex multiplier is an arrangement of four fixed-point multiply units which multiply two 2's complement 16-bit numbers and produce a single 16-bit result. The radix-4 Booth encoded multiplier is composed of six basic units. There is a 34-bit shift register, a modified ALU, a modified latch, a $2\times$ multiply unit, a Booth decoder and three controllers. These components are connected as shown in Figure 4-13.

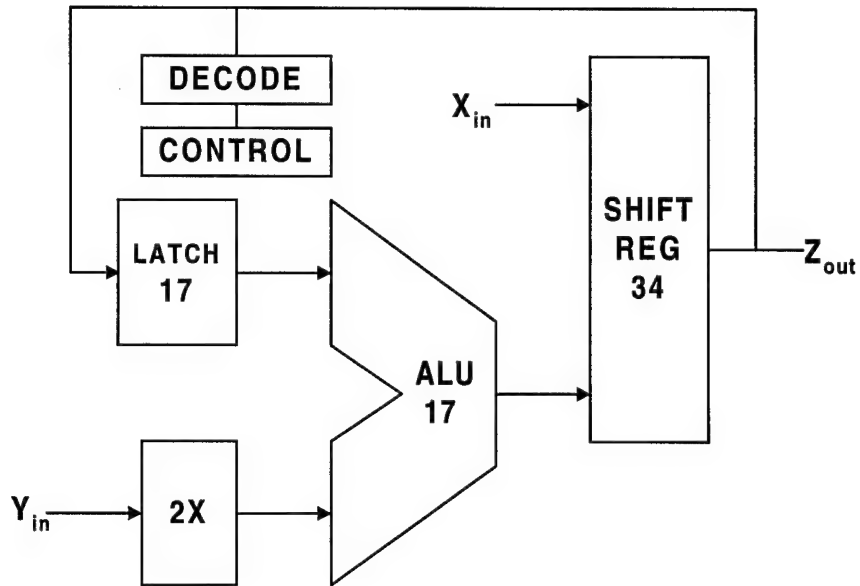


Figure 4-13. Booth Multiplier Block Diagram

4.3.3.1 34-bit Shift Register

The 34-bit shift register is composed of 20 resettable and 14 non-resettable D flip-flops. Additional logic was added to allow the reset or loading of the top 17 bits and loading of the lowest 17-bits. A shift signal causes the register to do an arithmetic shift (with MSB sign extension) right by two. The resettable D flip-flops are used to initialize the multiplier to a known state upon reset. They also zero the value of the top 17 bits for each new multiply. Figure 4-14 shows the interconnection of a few D flip-flops, as the entire schematic is too large to show here.

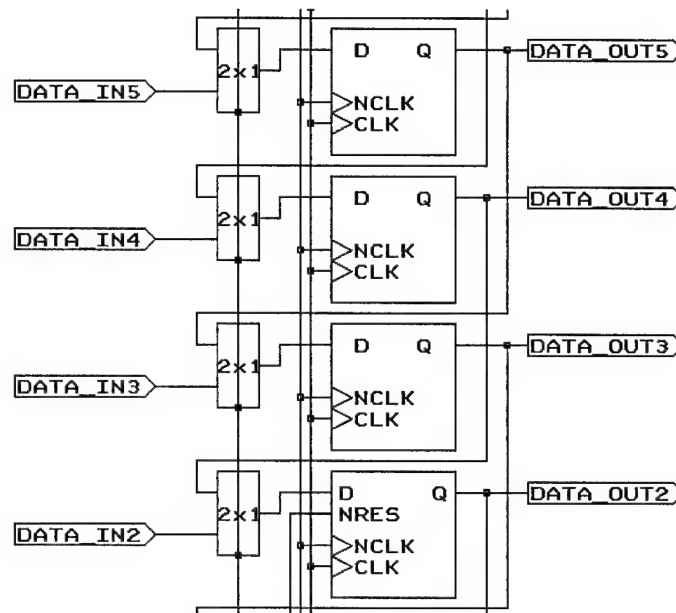


Figure 4-14. 34-Bit Shift Register Block Diagram

4.3.3.2 Booth Decoder

A Booth decoder was designed to implement the radix-4 Booth algorithm. It is a logic block that asserts 2X, ADDSUB and SHIFTONLY signals by looking at the least significant three bits of the 34-bit shift register. The truth table for the radix-4 algorithm is shown in Table 4-3 and logic is shown in Figure 4-15 [35].

Table 4-3. Radix-4 Booth Algorithm

Bit 2	Bit 1	Bit 0	Operation
0	0	0	Shift only
0	0	1	Add 1 x multiplicand, shift
0	1	0	Add 1 x multiplicand, shift
0	1	1	Add 2 x multiplicand, shift
1	0	0	Subtract 2 x multiplicand, shift
1	0	1	Subtract 1 x multiplicand, shift
1	1	0	Subtract 1 x multiplicand, shift
1	1	1	Shift only

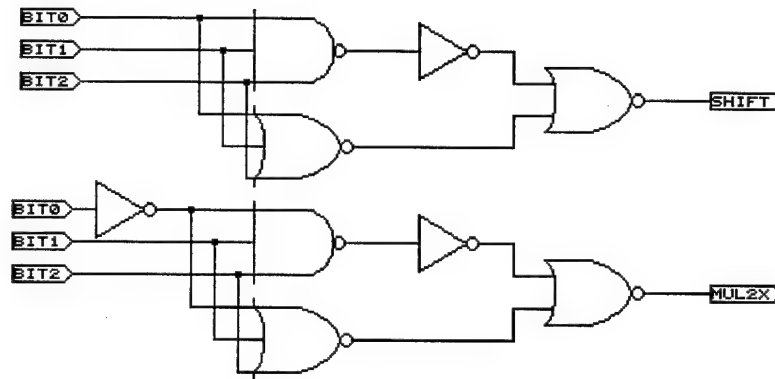


Figure 4-15. Radix-4 Booth Decoder

The MUL2X signal controls the operation of the 2X multiply. The ADDSUB signal, which goes to the ALU, is simply Bit 2. The SHIFT signal serves as a masking signal to the controllers in a SHIFT only condition (000 or 111) to simplify the controller design. It was found in the design process that without masking the control signals in this case, the controllers would have to handle this exception, which increases the size of the controllers.

4.3.3.3 Modified ALU

The basic asynchronous ALU16 used in the FFT-4 was modified for use in the multiplier. An additional stage was added to make the adder a 17-bit adder (ALU17). This additional bit was needed to handle sign extensions, which occurs in Booth multiplication.

4.3.3.4 Modified Latch

Similar to the modified ALU in the previous section, the LATCH16 used in the FFT-4 was extended by one bit to make a LATCH17 (stores 17-bits). This was necessary to latch the data from the upper 17 bits of the shift register.

4.3.3.5 2X Multiply Unit

This unit executes a multiply by 2 depending on the output of the Booth decoder. The multiply by two is done through 17 muxes with a wired sign extension.

4.3.3.6 Multiply Control

The multiply control is divided into three simple control blocks. The top controller interfaces with the external MULTREQ and MULTACK signals. From these external signals, the two remaining controllers are driven to complete the Booth multiplication. In the case of the “shift only” condition, the ALU operation of the controller is masked by minimum delay elements to allow for a quick shift by two. The AFSM state tables are shown in Appendix C.

4.4 Constant Banks

There are three constant banks that store the values of the W_{16} matrix presented in Equation 3-2. Each bank produces four constants. Due to the symmetry of FFTs, many of the constants are the same or are the negative of other values. The values in the constant banks are listed in Table 4-4, 4-5, and 4-6. The decimal values shown are the closest representation of the irrational values given 12-bit precision.

The constant banks were implemented with combinational logic controlled by the two-bit Johnson counter shown in Figure 2-6. The COUNT signal is fed by the ACKOUT signal of the FFT-4 that supplies the complex multiplier data. When the first number to be multiplied comes through, the counter has a value of 00. The counter value determines what constant will be fed into the complex multiplier. Figure 4-16 shows the

logic for determining the constants fed into CPLXMULT1, CPLXMULT2, and CPLXMULT3, respectively. The output values represented as A through K and MULBY1 are wired to the inputs of the complex multiplier to form the appropriate constant values. Logic was not necessary to represent all 16 combinations, as 10 combinations were enough to express the constants.

Table 4-4. Constant Bank One Values

Constant	Binary Equivalent		Decimal Equivalent	
	Real	Imaginary	Real	Imaginary
1	1000	0000	1.000000000000	0.000000000000
W^2	0B50	F4B0	0.707031250000	-0.707031250000
W	0EC8	F9E1	0.923828125000	-0.382568359375
W^3	061F	F138	0.382568359375	-0.923828125000

Table 4-5. Constant Bank Two Values

Constant	Binary Equivalent		Decimal Equivalent	
	Real	Imaginary	Real	Imaginary
1	1000	0000	1.000000000000	0.000000000000
$-j$	0000	F000	0.000000000000	-1.000000000000
W^2	0B50	F4B0	0.707031250000	-0.707031250000
$-jW^2$	F4B0	F4B0	-0.707031250000	-0.707031250000

Table 4-6. Constant Bank Three Values

Constant	Binary Equivalent		Decimal Equivalent	
	Real	Imaginary	Real	Imaginary
1	1000	0000	1.000000000000	0.000000000000
$-jW^2$	F4B0	F4B0	-0.707031250000	-0.707031250000
W^3	061F	F138	0.382568359375	-0.923828125000
$-W$	F138	061F	-0.923828125000	0.382568359375

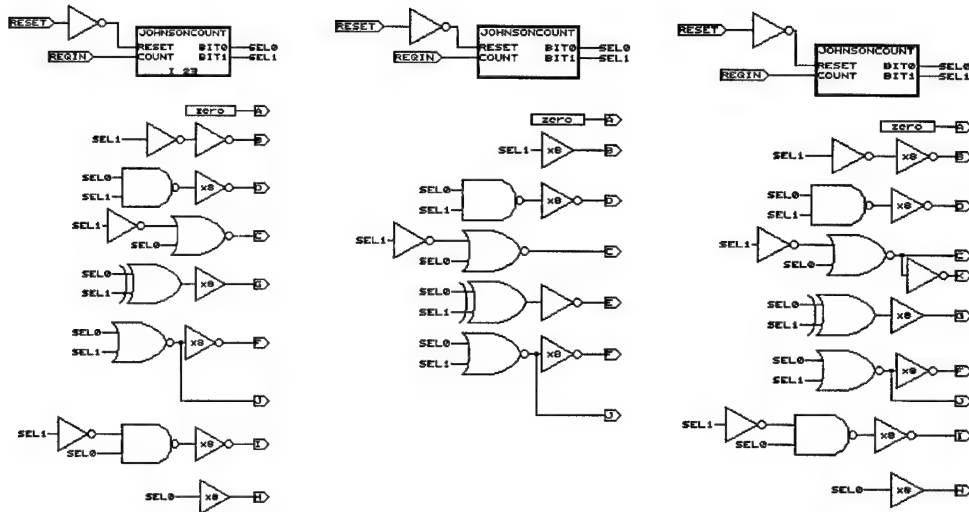


Figure 4-16. Constant Banks

4.5 Decimator

The decimator is a controller which interfaces the external FFT-16 REQIN/ACKIN to the first stage FFT-4 input request signals. Initially, an AFSM was synthesized from 3D that handled the control signals. The AFSM was discarded for a more simple design that uses a two bit Johnson counter as shown in Figure 4-17. The COUNT signal of the counter is fed by the ACKIN signal. Each full ACKIN transition increments the count after every four-cycle handshake. The count controls the MUX4X1 and selector circuit which enables the passing of the FFT-16 REQIN/ACKIN signals to the appropriate FFT-4. Data is passed to the input state FFT-4s through a shared bus corrected for fanout.

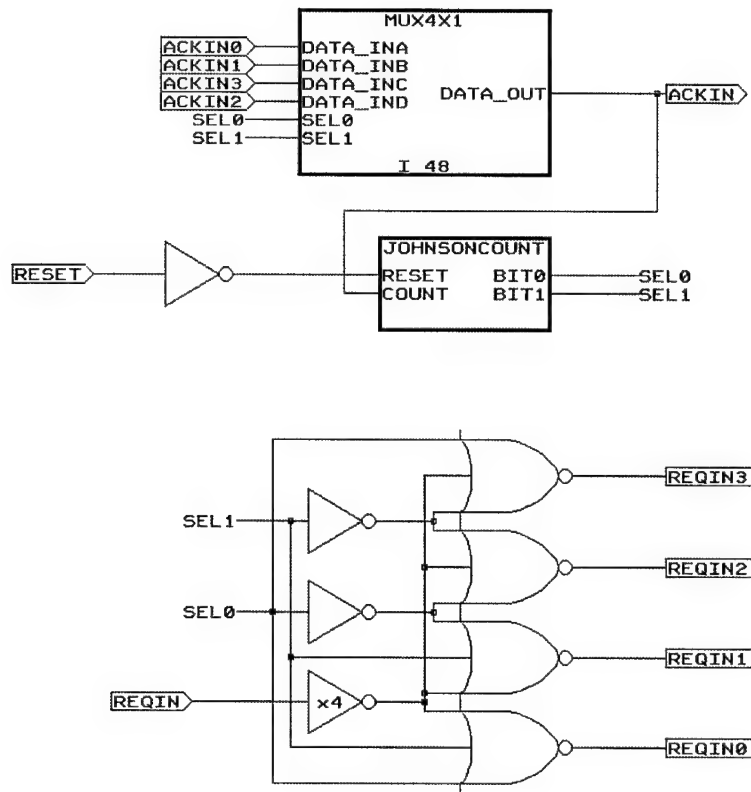


Figure 4-17. Decimator Gate-Level Schematic

4.6 Expander

Similar to the decimator, the expander interfaces the output stage FFT-4s output request signals to the FFT-16 output request signals. The COUNT signal of the two-bit Johnson counter is fed by the FFT-16 ACKOUT signal. The counter bits control a MUX4X1 and selector circuit which connects the correct output stage REQOUT/ACKOUT pair to the FFT-16 REQOUT/ACKOUT external signals. The SEL bits also control an output mux which routes the data from the appropriate output stage FFT-4 to the output. The expander schematic is shown in Figure 4-18.

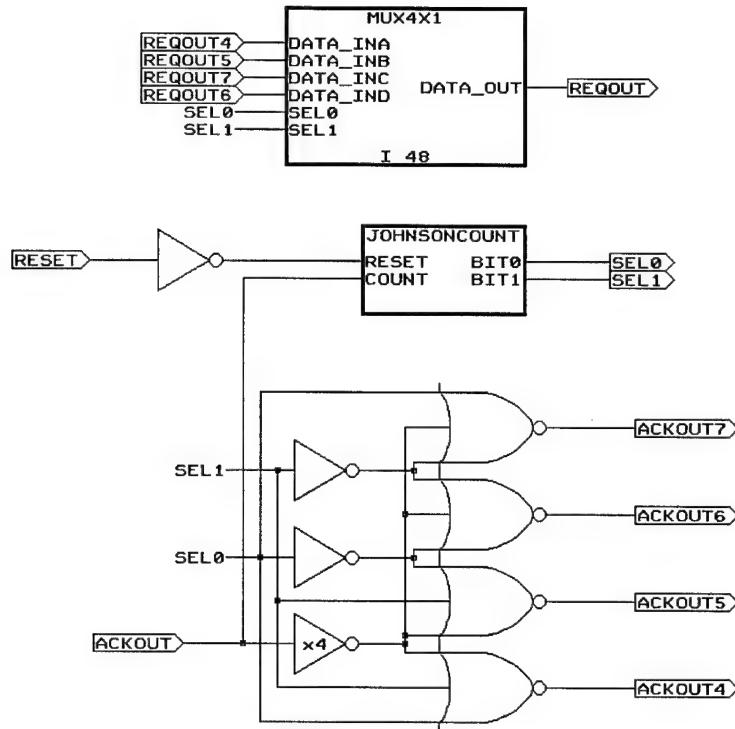


Figure 4-18. Expander Gate-Level Schematic

4.7 “Crossbar”

The “crossbar” handles the interconnection between the input stage output request signals and the output stage input request signals. The crossbar collects one data word from each input stage FFT-4 (0 through 3) and sends them in order to the first output stage FFT-4 (4). The cycle is repeated for the remaining output stage FFT-4s (5 through 7).

The first component designed for the crossbar was a divide by four circuit as shown in Figure 4-19. This divider was used to handle the difference in sampling frequency per FFT-4 between the input stage and output stage. The divider signaled the

change in routing signals to the output stage FFT-4s once for every four input stage FFT-4 output requests.

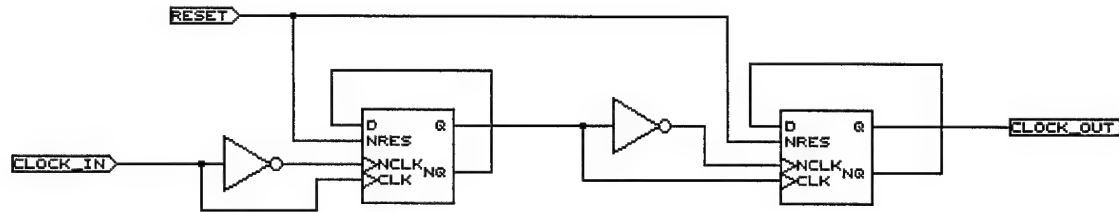


Figure 4-19. Divide By Four Schematic

A control circuit similar to an expander control circuit is used to funnel down the input stage control signals to a single REQOUT/ACKOUT pair. This expander also drives four write enable lines that are connected to tri-state buffers that control which FFT-4 is allowed to write its post-multiplied data to a shared bus. The output of the divide by four block, which is driven by the ACKOUT signal of the re-used expander, is fed into another two-bit Johnson counter which facilitates the four value input into each output stage FFT-4.

Because the outputs of the FFT-4 appear out of order, the crossbar is used to correct the order of the data by switching the FFT-4 (5) and FFT-4 (6) request lines. This eliminates the need to correct the output sequence from the input stage FFT-4s. The next section covers the reorder register, which is used to correct the output sequence of the output stage FFT-4s. The crossbar schematic is shown in Figure 4-20.

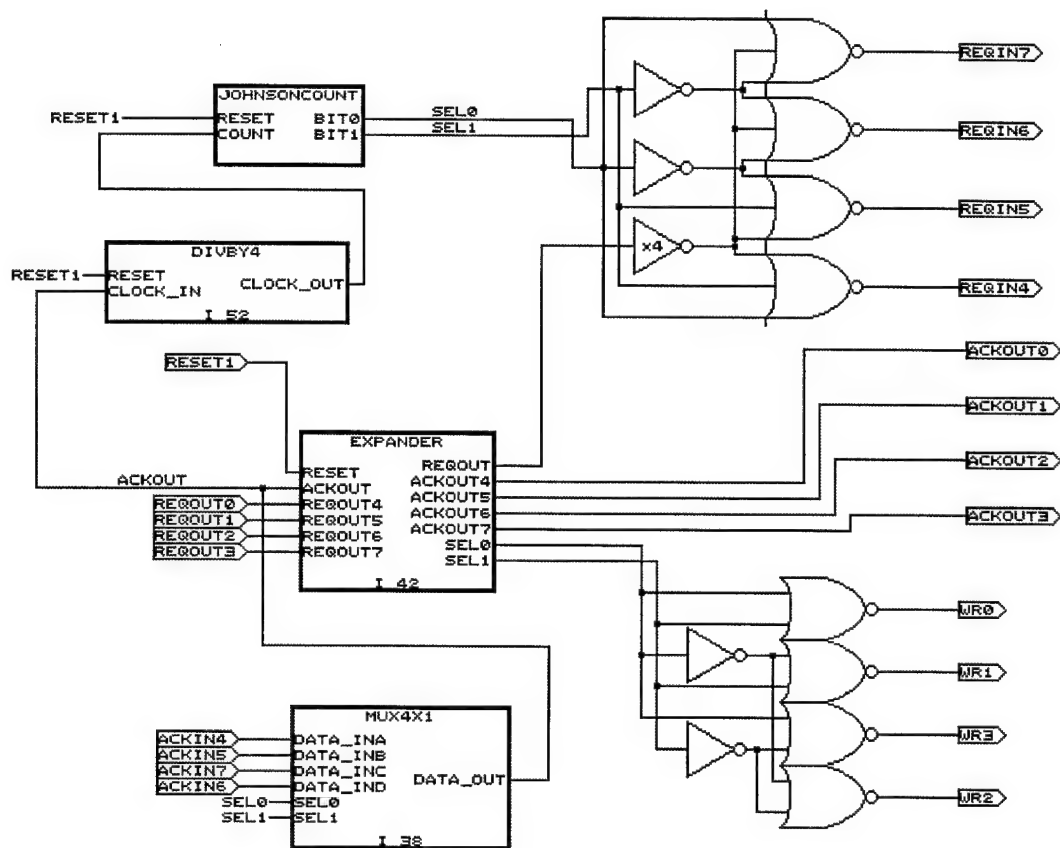


Figure 4-20. Crossbar Schematic

4.8 Reorder Register

The reorder register is a necessary component to correctly order the output of the FFT-4s on the output stage. As mentioned previously, the crossbar handles the output order sequence problem of the input stage FFT-4s. The reorder register is simply a pair of registers and muxes coupled to a small controller. The controller allows the first data word ($X(0)$) to pass straight through. The second REQIN latches the input data ($X(2)$), but does not give a REQOUT signal. The third REQIN along with the input data ($X(1)$) is

passed straight through as in the first case. The fourth REQIN triggers the mux to output the value in the register, which is $X(2)$. The controller then requests that the data on the input bus, $X(3)$ be passed through. This way the final output sequence of the FFT-4 is changed to $X(0), X(1), X(2), X(3)$. The component level schematic is shown in Figure 4-21.

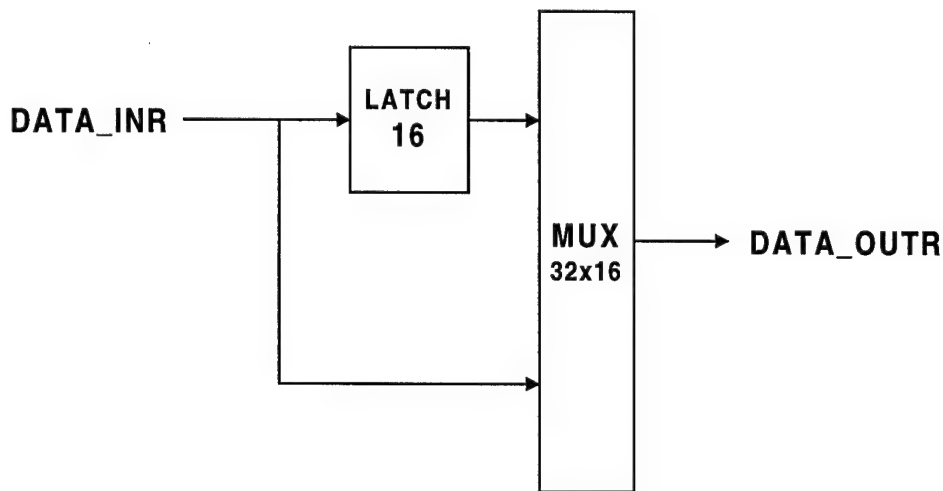


Figure 4-21. Reorder Register Schematic

4.9 Design Implementation Conclusion

Each major component was described in detail in this chapter, along with other designs considered for each component. The FFT-4, complex multiplier, decimator, and expander all work together to achieve the top-level FFT-16 functionality as shown in Figure 4-2. A reorder register was added to correct the output order of the FFT-4. The next chapter analyzes the components at all levels of simulation.

5. Results

5.1 *FFT-4 Test Chip*

The FFT-4 test chip did not return from fabrication in time for the test results to be included in this thesis. The purpose of the test chip was to validate that the asynchronous design methodologies chosen and the radiation tolerant library performed as expected. The FFT-4 test chip will be evaluated at a later date to determine if any corrections need to be made to the design of the FFT-16 before it is fabricated.

5.2 *Simulation Results*

This section presents the simulation results of each major component and the top-level FFT-16 design. VHDL simulation was used to verify proper operation and derive rough estimates of component timing information. Back-annotated timing data obtained from HSPICE simulations of each logic gate was used to describe gate delays to the VHDL simulator. IRSIM was used to simulate the performance of the layout and verify proper operation [13]. IRSIM is a mixed-mode simulator, which analyzes circuit extraction data and gives a good functionality check as well as more realistic timing information. Finally, the HSPICE results give the most accurate results possible, including power information, which is not available from either VHDL or IRSIM.

High-efficiency and performance were the end goals for the FFT-16. However, correct circuit operation and more importantly, correct results, was an implied

requirement. To demonstrate that the FFT-16 did indeed correctly transform a time domain sequence into a frequency domain sequence, multiple test cases were used for validation. One simple test case is the input of an impulse function. Mathcad [36] was used to calculate the expected output stream. The same impulse function was given to the FFT-16 and simulated in VHDL and IRSIM. Both VHDL and IRSIM produced the same results, as shown in Figures D-27 and D-28. Table 5-1 shows the input time-domain sequence $x(n)$. Table 5-2, Figure 5-1 and Figure 5-2 show the comparison of the output frequency-domain sequence $X(m)$. This test sequence clearly demonstrates that the FFT-16 does produce the correct $X(m)$ for a given $x(n)$ with a small error generated due to the selected fixed-point data format.

Table 5-1. Impulse Function Input Sequence

n	$x(n)$
0	0
1	1
2 through 15	0

Table 5-2. Mathcad vs. Simulation Results

m	Mathcad Results		Simulation Results		Maximum Error (%)
	$\text{Re}\{X(m)\}$	$\text{Im}\{X(m)\}$	$\text{Re}\{X(m)\}$	$\text{Im}\{X(m)\}$	
0	1.000000000000	0.000000000000	1.000000000000	0.000000000000	0.00000
1	0.923879532511	-0.382683432365	0.923828125000	-0.382568359375	-0.00556
2	0.707106781187	-0.707106781187	0.707031250000	-0.707031250000	-0.01068
3	0.382683432365	-0.923879532511	0.382568359375	-0.923828125000	-0.00556
4	0.000000000000	-1.000000000000	0.000000000000	-1.000000000000	0.00000
5	-0.382683432365	-0.923879532511	-0.382568359375	-0.923828125000	-0.00556
6	-0.707106781187	-0.707106781187	-0.707031250000	-0.707031250000	-0.01068
7	-0.923879532511	-0.382683432365	-0.923828125000	-0.382568359375	-0.00556
8	-1.000000000000	0.000000000000	-1.000000000000	0.000000000000	0.00000
9	-0.923879532511	0.382683432365	-0.923828125000	0.382568359375	-0.00556
10	-0.707106781187	0.707106781187	-0.707031250000	0.707031250000	-0.01068
11	-0.382683432365	0.923879532511	-0.382568359375	0.923828125000	-0.00556
12	0.000000000000	1.000000000000	0.000000000000	1.000000000000	0.00000
13	0.382683432365	0.923879532511	0.382568359375	0.923828125000	-0.00556
14	0.707106781187	0.707106781187	0.707031250000	0.707031250000	-0.01068
15	0.923879532511	0.382683432365	0.923828125000	0.382568359375	-0.00556

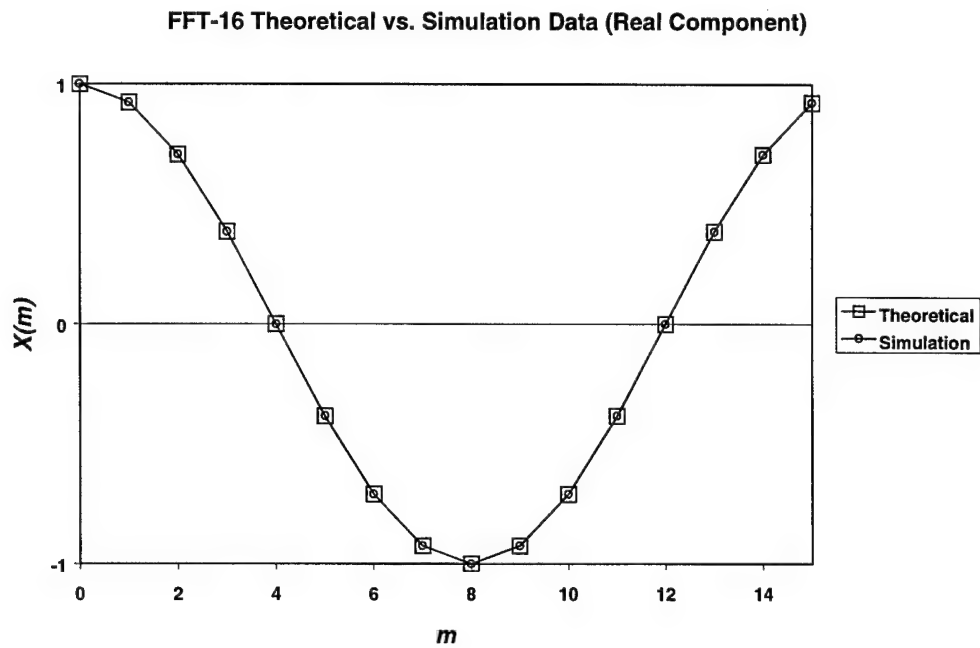


Figure 5-1. Theoretical vs. Simulation Data (Real Component)

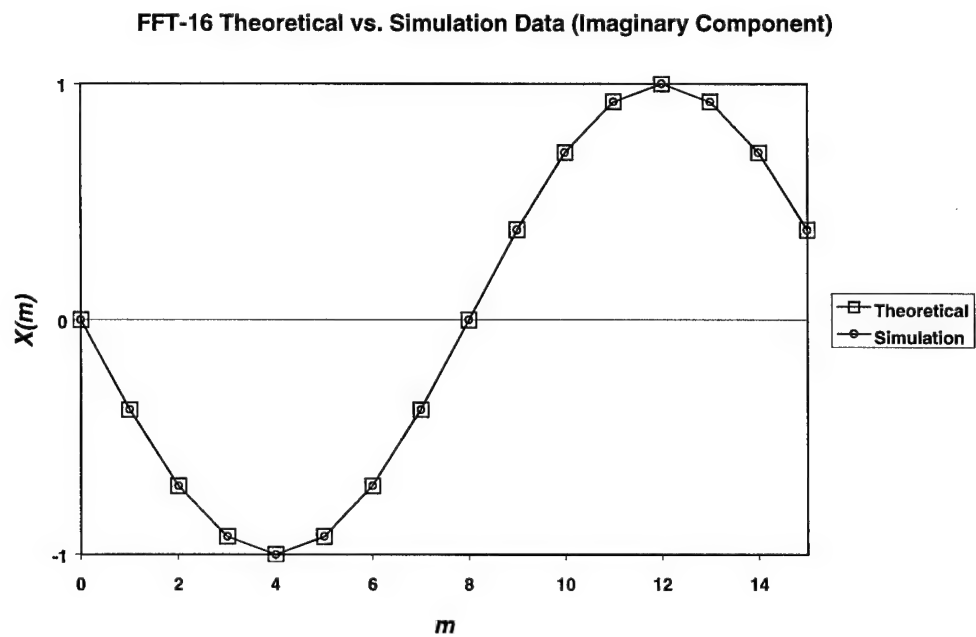


Figure 5-2. Theoretical vs. Simulation Data (Imaginary Component)

Table 5-3 gives a summary of the FFT-16 design statistics and the performance obtained through simulation. Component latencies are given for the worst-case propagation time through the component. The throughput is given for the FFT-4 and FFT-16. The energy measurement for each component is the energy required for that component in a single FFT-16 calculation. The computational resources were not available to run a top-level FFT-16 HSPICE simulation, so the FFT-16 energy requirement was determined by the sum of the energy required for each component multiplied by the number of instances. This energy measurement gives an efficiency of 28 nJ/Unit Transform for the FFT-16. Appendix D contains the simulation waveforms used to obtain the data in Table 5-3.

Table 5-3. FFT-16 Design and Simulation Results

Component	Design Statistics		Component Latency (ns)			Throughput (ns)	Energy (nJ)
	Area (μm^2)	Transistors	VHDL	IRSIM	HSPICE	IRSIM	HSPICE
Decimator	41,000	271	1.0	1.0	1.0	-	0.26
Crossbar	110,000	1,143	1.0	1.0	2.0	-	3.0
Expander	43,000	271	1.0	1.0	1.0	-	0.25
Reorder Reg.	340,000	2,014	13	16	9.0	-	1.4
FFT-4	3,400,000	20,340	80	180	180	180	17
Complex Mult.	5,300,000	33,149	85	110	134	-	98
FFT-16	45,000,000	271,908	585	980	-	760	440

The area and transistor count is also given for each component and the top-level circuit. Because a two-metal channel router was used to route the circuits, the actual area that can be realized from a high performance three-metal over-the-cell router will reduce the area by 50%. A simulation of the re-routed circuit will also show a slight decrease in the energy consumption of the circuit due to the optimized routing which lowers the trace capacitance throughout the circuit.

The worst case throughput time of 760 ns and efficiency of 28 nJ/Unit transform of the FFT-16 must be extrapolated to an FFT-1024 for comparison with the FFT processors presented in Chapter Two. The throughput time of the FFT-1024 is derived by dividing the latency of the slowest block in the FFT-1024 (complex multiplier) by the decimation at the top level (64). This gives a throughput of 2 μ s. The efficiency is calculated by summing the energy of the components required for calculating a single point of an FFT-1024. This gives an extrapolated FFT-1024 efficiency of 120 nJ/Unit Transform. Table 2-2 is presented again here with the new FASST extrapolated data as Table 5-4.

Table 5-4 shows that the design produced in this thesis effort has roughly the same performance when compared to the previous thesis effort. However, the results of this research should be considered more reliable, as the results of the previous thesis were extrapolated from a 6-bit design.

In conclusion, the FASST architecture combined with asynchronous design and a radiation tolerant library indeed produced a design that is acceptable for space, as it offers two orders of magnitude improvement over typical space-based FFT processors in both throughput time and efficiency.

Table 5-4. Final Comparison of FFT Processors

Processor Name	Design Feature	Dataword Size & Type	Supply Voltage (V)	FFT-1024 throughput time (μ s)	Efficiency (nJ/Unit Transform)
C40	Space	32-Bit Floating Point	5.0	1298	5704
Spiffee	Low Power	20-bit Fixed Point	3.3	30	24.7
COBRA	Speed	23-bit Fixed Point	5.0	9.5	71.4
FASST (previous)	Asynchronous, space & low power	16-bit Fixed Point	3.3	10	120
FASST (new)	Asynchronous, space & low power	16-bit Fixed Point	3.3	2	120

6. *Summary and Conclusions*

6.1 *Summary*

This thesis has presented the steps taken to develop an energy-efficient high-performance asynchronous FFT-16 designed for space. Lessons learned from previous research were used as a starting point. New concepts and designs were developed. Finally, all of the components were integrated to achieve the top-level FFT-16 design and simulations were used to validate correct operation and to evaluate performance.

6.2 *Conclusions*

The results presented in Chapter Five clearly demonstrate that an asynchronous implementation of the FASST architecture can potentially facilitate the design of large-point FFT processors suitable for the space environment. The estimated throughput time of 2 μ s and efficiency of 120 nJ/Unit-Transform for an FFT-1024 offers an improvement of two orders of magnitude over existing space-based FFT processors. It is even more impressive to realize that this design is highly competitive with similar terrestrial designs! The significant efficiency and performance improvements are justification alone to continue this area of research and build larger point size FFT processors.

6.3 Lessons Learned

Throughout the course of the design process, many lessons were learned. Specifically, some valuable lessons were learned by using the MRC library and the MOSIS [37] service.

The use of the MRC library presented some interesting design challenges. The only available component of the library at the time of design was the physical layout of the cells. HSPICE simulation data of the cells and generic symbols were used to build a VHDL library for structural simulation with timing information. Fanout had to be manually compensated for throughout the design, as no library file was available for the Synopsys Design Analyzer, although one could have been developed. An interface also had to be built to the Lager Octtools layout tool for the place and route phase of the design.

Future design efforts with the radiation tolerant cells will benefit from the library files produced by MRC when they are available. The VLSI lab will also greatly benefit from using an advanced cell router to take advantage of the ability of the radiation tolerant cells to be densely packed.

An interesting lesson learned by using the MOSIS service was discovering the difference between technology layout rules and process specific fabrication rules [37]. Upon first submittal to the MOSIS service, the test chip design was rejected due to "oversized features on the contact layer." Discussing the problem with the MOSIS representative revealed that the HP 0.5 μm process requires that any feature on the contact layer (meaning poly contacts or metal vias) be *no larger* or *no smaller* than the

minimum size. The design rule check (DRC) feature of MAGIC [13] only checks if a feature is too small or too closely spaced, not if it is too large [13]. This is due to the fact that some fabrication processes take oversized contact features and break them up automatically into minimum sized contacts. Correcting the oversized contacts resulted in a successful submission.

6.4 *Recommendations for Future Research*

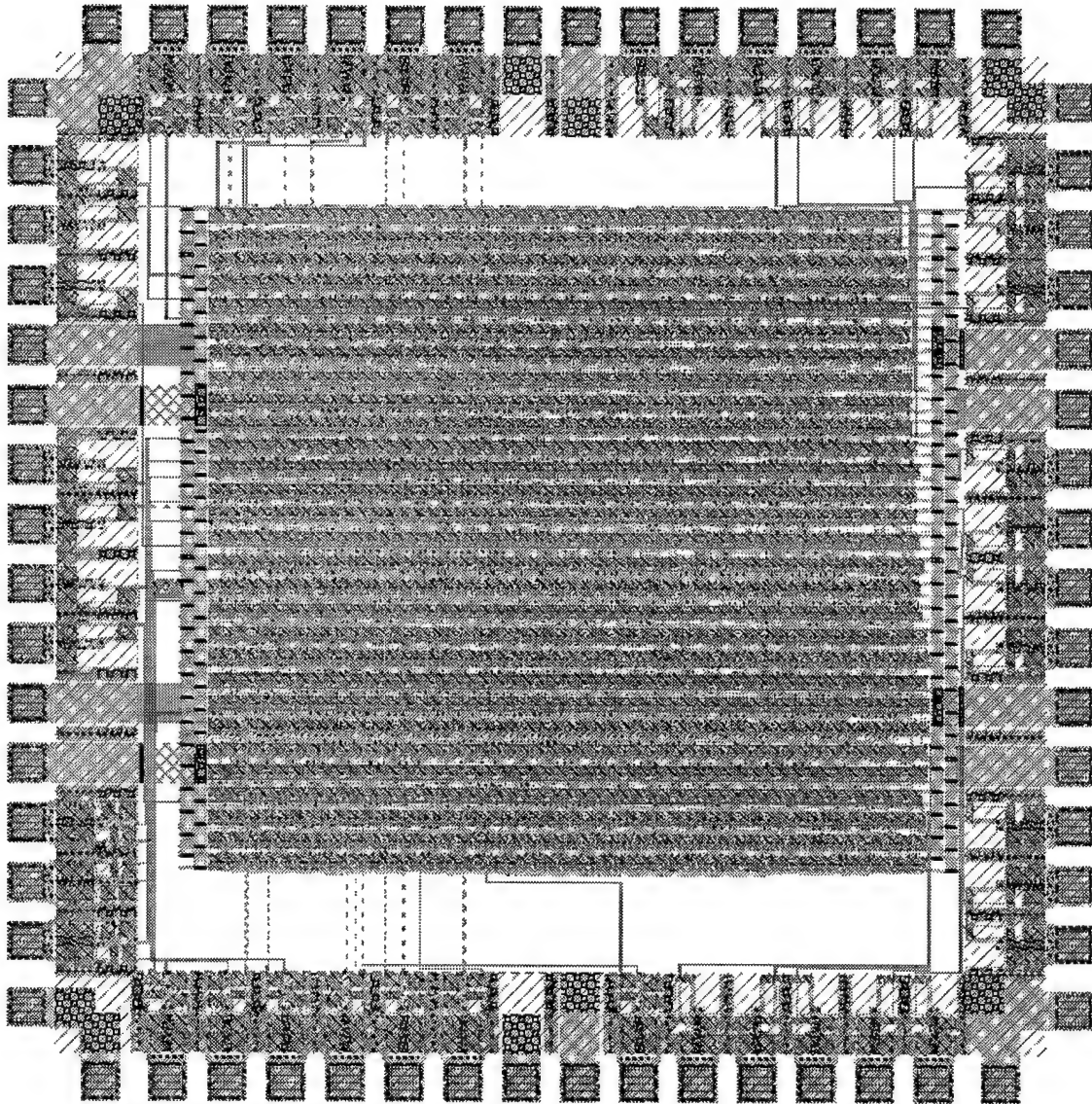
Clearly, the next step for research in this area is to implement larger point sizes with the FASST architecture. There is also a need to improve the asynchronous design flow.

The next logical point size to be developed is the FFT-256. With the FFT-16 building block developed in this research, a FASST FFT-256 can be developed with $N_1=N_2=16$. The complex multiplier design should be resized to meet the performance criteria of the FFT-256. New decimator, crossbar and expander elements would have to be developed also. The physical area of the FFT-256 will be much larger than the FFT-16 and will require the use of a high performance router to get the greatest cell density possible and reduce long metal trace issues present in larger designs.

An improved asynchronous design flow has been the goal of the growing asynchronous design community [3]. This field of research is wide open for new design concepts, implementations and tools. One facet of the design process that could be improved upon with the tools available in the AFIT VLSI lab is the interface with the 3D tool. A simple interface program could be developed to convert behavioral VHDL

written in a pre-defined format to the input file format of the 3D asynchronous tool. Then, from the output of the 3D tool, another translator program could be written to convert the output file (which is in positive logic) into a VHDL format. The Synopsys Design Analyzer or other such tool could then convert the positive logic VHDL into negative logic structural VHDL and correct for fanout. This design process was manually repeated throughout the work of this thesis. Such a set of interface programs would have been of great benefit and would have allowed rapid prototyping of new designs.

Appendix A. Layout of the Fabricated FFT-4 Design



Appendix B. FFT-4 IC Specification Sheet

Table B-1. FFT-4 Test Chip Specifications

Package Type:	PGA65
Supply Voltage:	3.3 VDC
Average Power (Simulated):	41.5 mW
Core Transistor Count:	21,951
Total Chip Area:	2934×2934 μm
Data Input Sequence:	A: $\text{Re}\{x(0)\}$ B: $\text{Re}\{x(2)\}$ A: $\text{Re}\{x(1)\}$ B: $\text{Re}\{x(3)\}$ A: $\text{Im}\{x(1)\}$ B: $\text{Im}\{x(3)\}$ A: $\text{Im}\{x(0)\}$ B: $\text{Im}\{x(2)\}$
Data Output Sequence:	$\text{Re}\{X(0)\}$ $\text{Re}\{X(2)\}$ $\text{Re}\{X(1)\}$ $\text{Re}\{X(3)\}$ $\text{Im}\{X(1)\}^*$ $\text{Im}\{X(3)\}^*$ $\text{Im}\{X(0)\}^*$ $\text{Im}\{X(2)\}^*$
*Note: Im values should be multiplied by (–1) to get the correct value.	

Testing Procedure:

1. Refer to Table B-2 for pin names.
2. Apply power, control and data connections to the chip.
3. Apply RESET signal with REQ_INA=0, REQ_INB=0, ACKOUT=0, BIDIR_SEL=1, TEST_SEL=00 and DATA_IN values set to zero until ACK_INA, ACK_INB and REQOUT all stabilize to zero.
4. Input data with alternating REQ_INA and REQ_INB signals, as shown in Table B-1. Leave REQ_IN signal high and keep data stable until appropriate ACK_IN signal is asserted then lower REQ_IN. Do not REQ_IN again until ACK_IN goes low.
5. When REQ_OUT goes high read data, then pulse ACK_OUT.
6. Use BIDIR_SEL=0 and desired TEST_SEL value to view test signals at any time during testing, except for when REQ_INA or REQ_INB is high. Refer to Table B-3 for viewable test signals.

Table B-2. Fabricated FFT-4 Pin List

PIN	SIGNAL IN	DATA OUT
1	PAD VDD	-
2	-	ACK_INB
3	-	DATA_OUT0
4	-	DATA_OUT2
5	-	DATA_OUT1
6	-	PAD_TEST_OUT
7	PAD_TEST_IN	-
8	CORE GND	-
9	CORE VDD	-
10	DATA_IN5	TEST_OUT2
11	DATA_IN4	TEST_OUT1
12	DATA_IN3	TEST_OUT5
13	DATA_IN2	TEST_OUT6
14	DATA_IN1	TEST_OUT8
15	DATA_IN0	TEST_OUT7
16	PAD GND	-
17	PAD GND	-
18	ACK_OUT	-
19	REQ_INB	-
20	TEST_SEL1	-
21	CORE VDD	-
22	CORE GND	-
23	TEST_SEL0	-
24	RESET	-
25	BIDIR_SEL (1= normal, 0= view test signals)	-
26	REQ_INA	-
27	CORE VDD	-
28	CORE GND	-
29	DATA_IN6	TEST_OUT4
30	DATA_IN7	TEST_OUT0
31	DATA_IN14	TEST_OUT3
32	PAD VDD	-
33	PAD VDD	-
34	DATA_IN15	TEST_OUT9
35	DATA_IN12	TEST_OUT10
36	DATA_IN13	TEST_OUT11
37	DATA_IN8	TEST_OUT14
38	DATA_IN11	TEST_OUT15
39	DATA_IN10	TEST_OUT13
40	PAD VDD	-
41	PAD GND	-
42	DATA_IN9	TEST_OUT12
43	-	DATA_OUT10
44	-	DATA_OUT9
45	-	DATA_OUT11
46	-	DATA_OUT8
47	-	DATA_OUT12
48	PAD GND	-
49	PAD GND	-
50	-	DATA_OUT13
51	-	DATA_OUT14
52	-	DATA_OUT15
53	CORE GND	-
54	CORE VDD	-
55	-	ACK_INA
56	-	DATA_OUT7
57	-	DATA_OUT6
58	-	DATA_OUT5
59	CORE GND	-
60	CORE VDD	-
61	-	DATA_OUT4
62	-	REQ_OUT
63	-	DATA_OUT3
64	PAD VDD	-

Table B-3. Test Signals

Test Pin	TEST_SEL	Internal Test Signal
TEST_OUT0	00	AREQ0
	01	AREQ2
	10	LREQ0
	11	REQOUTOP
TEST_OUT1	00	AACK0
	01	AREQ3
	10	LACK0
	11	ACKREQOUTOP
TEST_OUT2	00	AREQ1
	01	AREQ7
	10	LREQ1
	11	MUXSTATE
TEST_OUT3	00	AACK1
	01	AACK2
	10	LACK1
	11	MREQ
TEST_OUT4	00	AREQ4
	01	AACK3
	10	LREQ2
	11	MACK
TEST_OUT5	00	AACK40
	01	AACK7
	10	LACK2
	11	AREQSTATE
TEST_OUT6	00	ADD_SUB0
	01	ADD_SUB2
	10	LREQ3
	11	AACKSTATE
TEST_OUT7	00	ADD_SUB1
	01	ADD_SUB3
	10	LACK3
	11	SEL0
TEST_OUT8	00	ADD_SUB4
	01	ADD_SUB7
	10	REQREG
	11	SEL1
TEST_OUT9	00	BUS0A15
	01	BUS2A15
	10	ACKREG
	11	ADDSTATE
TEST_OUT10	00	BUS0B15
	01	BUS2B15
	10	GETREG
	11	AREQ4P
TEST_OUT11	00	BUS1A15
	01	BUS3A15
	10	AREQ5
	11	AREQ5P
TEST_OUT12	00	BUS1B15
	01	BUS3B15
	10	AACK5
	11	AREQ6P
TEST_OUT13	00	AC15
	01	FH15
	10	AREQ6
	11	AREQ7P
TEST_OUT14	00	EG15
	01	BD15
	10	AACK6
	11	RESETCIN
TEST_OUT15	00	BUS415
	01	BUS715
	10	ADD_SUB5
	11	ADD_SUB6

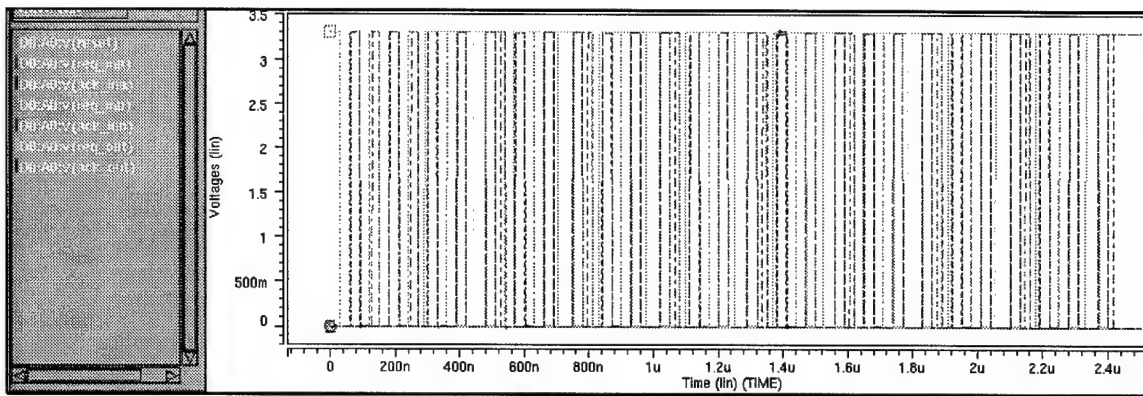


Figure B-1. FFT-4 Test Chip HSPICE Simulation of Timing

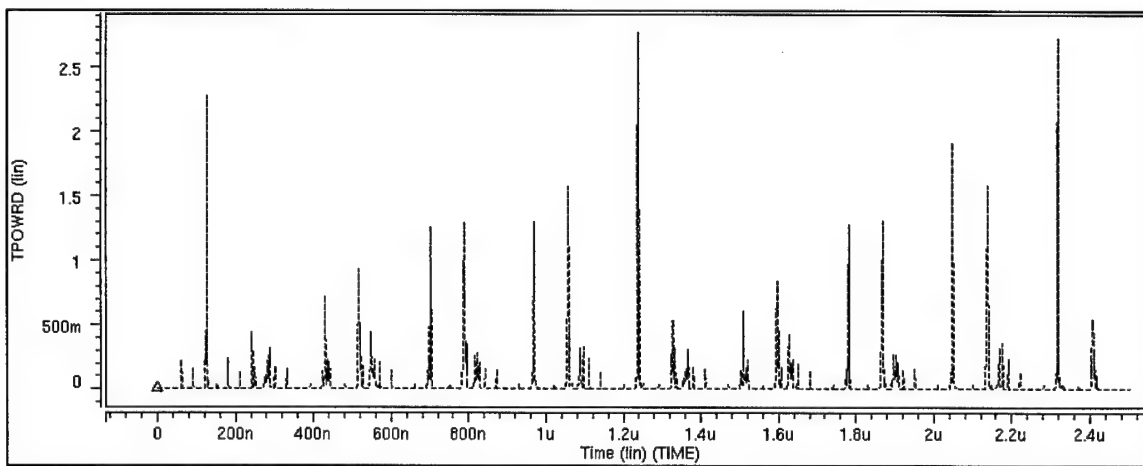


Figure B-2. FFT-4 Test Chip Power

Appendix C. AFSM Descriptions

Table C-1. FFT-4 CONTROLIN AFSM

```

input reqin 0
input reqdata 0
input lack 0

output ackin 0
output lreq 0
output reqdataout 0

;;; Current Next Input Burst | Output Burst
;;; State State |
0 1 reqin+ reqdata+ | lreq+
1 2 lack+ | ackin+
2 3 reqin- | ackin-
3 4 reqin+ | lreq-
4 5 lack- | ackin+
5 6 reqin- | ackin-
6 7 reqin+ | lreq+
7 8 lack+ | ackin+
8 9 reqin- | ackin- reqdataout+
9 10 reqin+ reqdata- | lreq-
10 11 lack- | ackin+
11 0 reqin- | ackin- reqdataout-

;;;-----
;;; 3D Synthesized Equations:
;;;-----

;;; ackin =
;;; reqin lack' reqdataout +
;;; reqin lack zzz01 +
;;; reqin lack reqdataout' zzz00 +
;;; reqin lack' zzz00' zzz01'

;;; lreq =
;;; reqin' lreq +
;;; ackin lreq +
;;; reqdata reqdataout +
;;; reqin reqdataout' zzz00 +
;;; reqin reqdata zzz01

;;; reqdataout =
;;; reqin reqdataout +
;;; lack reqdataout +
;;; reqin' lreq zzz00

;;; zzz00 =
;;; reqin zzz00 +
;;; reqdata zzz00 +
;;; lreq zzz00 +
;;; reqin' reqdata lreq' zzz01'

;;; zzz01 =
;;; reqin zzz01 +
;;; lack' zzz01 +
;;; reqin' reqdata' lreq'

```

Table C-2. FFT-4 CONTROLGETREG AFSM Description

input	getreg	0		
input	lack0	0		
input	lack1	0		
input	lack2	0		
input	lack3	0		
output	ackreg	0		
output	lreq0	0		
output	lreq1	0		
output	lreq2	0		
output	lreq3	0		
;;;	Current	Next	Input Burst	Output Burst
;;;	State	State		
	0	1	getreg+	lreq0+
	1	2	lack0+	lreq0-
	2	3	lack0-	ackreg+
	3	4	getreg-	lreq1+
	4	5	lack1+	lreq1-
	5	6	lack1-	ackreg-
	6	7	getreg+	lreq2+
	7	8	lack2+	lreq2-
	8	9	lack2-	ackreg+
	9	10	getreg-	lreq3+
	10	11	lack3+	lreq3-
	11	0	lack3-	ackreg-

;;; 3D Synthesized Equations:				

;;; ackreg =				
;;; lack1 +				
;;; lack3 +				
;;; lack0' lack2' zzz01				
;;; lreq0 =				
;;; getreg lack0' zzz00' zzz01'				
;;; lreq1 =				
;;; getreg' lack1' lack3' zzz00' zzz01				
;;; lreq2 =				
;;; getreg lack2' zzz00 zzz01'				
;;; lreq3 =				
;;; getreg' lack1' lack3' zzz00 zzz01				
;;; zzz00 =				
;;; lack1 +				
;;; lack3' zzz00				
;;; zzz01 =				
;;; lack0 +				
;;; lack2 +				
;;; lack1' lack3' zzz01				

Table C-3. FFT-4 CONTROLOUTA AFSM Description

```
input reqdataout 0
input done       0
input reset      0
```

```
output areq03    0
output areq12    0
output reqdata   0
output sel       0
```

Current State	Next State	Input Burst	Output Burst
0	1	reset+	reqdata+
1	2	reqdataout+	areq03+ reqdata-
2	3	reqdataout-	areq12+
3	4	done+	areq03- areq12- sel+
4	5	done-	areq03+ areq12+
5	6	done+	areq03- areq12- sel-
6	1	done-	reqdata+

```
;;;-----
;;; 3D Synthesized Equations:
;;;-----

;;; areq03 =
;;; reqdataout +
;;; done' areq03 +
;;; done' sel

;;; areq12 =
;;; done' sel +
;;; reqdataout' done' areq03

;;; reqdata =
;;; reqdataout' done' reset areq03' sel'

;;; sel =
;;; done' sel +
;;; done zzz00' +
;;; sel zzz00'

;;; zzz00 =
;;; done' sel +
;;; done zzz00 +
;;; sel zzz00
```

Table C-4. FFT-4 CONTROLOUTB AFSM Description

input	aacks1	0		
input	aacks2	0		
input	ackout	0		
output	areqs2	0		
output	reqout	0		
output	addsubs2	0		
output	done	0		
;;; Current	Next	Input Burst		Output Burst
;;; State	State			
	0	1	aacks1+	areqs2+
	1	2	aacks2+	reqout+
	2	3	ackout+	reqout-
	3	4	ackout-	areqs2- addsubs2+
	4	5	aacks2-	areqs2+
	5	6	aacks2+	reqout+
	6	7	ackout+	reqout-
	7	8	ackout-	areqs2- addsubs2-
	8	9	aacks2-	done+
	9	0	aacks1-	done-

;;; 3D Synthesized Equations:				

;;; areqs2 =				
;;; ackout +				
;;; aacks2' addsubs2 +				
;;; aacks1 zzz00				
;;; reqout =				
;;; aacks2 ackout' zzz00				
;;; addsubs2 =				
;;; ackout addsubs2 +				
;;; addsubs2 zzz01' +				
;;; aacks2 ackout' zzz00' zzz01'				
;;; done =				
;;; aacks1 aacks2' addsubs2' zzz00'				
;;; zzz00 =				
;;; aacks1' +				
;;; aacks2' addsubs2 +				
;;; ackout' zzz00				
;;; zzz01 =				
;;; ackout addsubs2 +				
;;; aacks2 zzz01				

Table C-5. Booth Multiplier CONTROLMULT AFSM Description

input	multreq	0		
input	lackA	0		
input	boothack	0		
output	multack	0		
output	lreqA	0		
output	boothreq	0		
output	zero	0		
;;;	Current	Next	Input Burst	Output Burst
;;;	State	State		
	0	1	multreq+	lreqA+ zero+
	1	2	lackA+	lreqA- zero-
	2	3	lackA-	boothreq+
	3	4	boothack+	boothreq-
	4	5	boothack-	multack+
	5	0	multreq-	multack-

;;; 3D Synthesized Equations:				

;;; multack =				
;;; multreq boothack' zzz00				
;;; lreqA =				
;;; multreq lackA' boothack' zzz00' zzz01'				
;;; boothreq =				
;;; lackA' boothack' zzz01				
;;; zero =				
;;; multreq lackA' boothack' zzz00' zzz01'				
;;; zzz00 =				
;;; boothack +				
;;; multreq zzz00				
;;; zzz01 =				
;;; lackA +				
;;; boothack' zzz01				

Table C-6. Booth Multiplier CONTROLCALC AFSM Description

```
input  calcreq  0
input  alulack  0
input  aack     0
input  lackB    0
input  sack     0
```

```
output alulreq  0
output areq     0
output lreqB    0
output sreq     0
output calcack  0
```

Current State	Next State	Input Burst	Output Burst
0	1	calcreq+	alulreq+
1	2	alulack+	alulreq-
2	3	alulack-	areq+
3	4	aack+	lreqB+
4	5	lackB+	lreqB-
5	6	lackB-	areq-
6	7	aack-	sreq+
7	8	sack+	sreq-
8	9	sack-	calcack+
9	0	calcreq-	calcack-

```
;;;-----
;;; 3D Synthesized Equations:
;;;-----
```

```
;;; alulreq =
;;; calcreq alulack' areq' zzz00' zzz01'
```

```
;;; areq =
;;; lackB +
;;; calcreq alulack' zzz00' zzz01
```

```
;;; lreqB =
;;; aack lackB' zzz01
```

```
;;; sreq =
;;; calcreq aack' sack' zzz00 zzz01'
```

```
;;; calcack =
;;; calcreq sack' areq' zzz00 zzz01
```

```
;;; zzz00 =
;;; lackB +
;;; calcreq zzz00
```

```
;;; zzz01 =
;;; alulack +
;;; sack +
;;; calcreq lackB' zzz01
```

Table C-7. Booth Multiplier CONTROLBOOTH AFSM Description

```

input boothreq 0
input calcack 0

output boothack 0
output calcreq 0

;;; Current Next      Input Burst      | Output Burst
;;; State  State
    0          1      boothreq+      | calcreq+
    1          2      calcack+      | calcreq-
    2          3      calcack-      | calcreq+
    3          4      calcack+      | calcreq-
    4          5      calcack-      | calcreq+
    5          6      calcack+      | calcreq-
    6          7      calcack-      | calcreq+
    7          8      calcack+      | calcreq-
    8          9      calcack-      | calcreq+
    9         10      calcack+      | calcreq-
   10         11      calcack-      | calcreq+
   11         12      calcack+      | calcreq-
   12         13      calcack-      | calcreq+
   13         14      calcack+      | calcreq-
   14         15      calcack-      | calcreq+
   15         16      calcack+      | calcreq-
   16         17      calcack-      | boothack+
   17          0      boothreq-      | boothack-

```

```

;;;-----
;;; 3D Synthesized Equations:
;;;-----

```

```

;;; boothack =
;;; boothreq calcack' zzz12 zzz13

```

```

;;; calcreq =
;;; calcack' zzz13' +
;;; boothreq calcack' zzz12'

```

```

;;; zzz00 =
;;; boothreq zzz00 +
;;; calcack' zzz13'

```

```

;;; zzz01 =
;;; calcack zzz00 +
;;; calcack' zzz01 +
;;; zzz00 zzz01

```

```

;;; zzz02 =
;;; boothreq zzz02 +
;;; calcack' zzz01 zzz13'

```

```

;;; zzz03 =
;;; calcack zzz02 +
;;; calcack' zzz03 +
;;; zzz02 zzz03

```

```

;;; zzz04 =
;;; boothreq zzz04 +
;;; calcack' zzz03 zzz13'

```

```

;;; zzz05 =
;;; calcack zzz04 +
;;; calcack' zzz05 +
;;; zzz04 zzz05

```

```

;;; zzz06 =
;;; boothreq zzz06 +
;;; calcack' zzz05 zzz13'

```

```

;;; zzz07 =
;;; calcack zzz06 +
;;; calcack' zzz07 +
;;; zzz06 zzz07

```

```

;;; zzz08 =
;;; boothreq zzz08 +
;;; calcack' zzz07 zzz13'

```

```

;;; zzz09 =
;;; calcack zzz08 +
;;; calcack' zzz09 +
;;; zzz08 zzz09

```

```

;;; zzz10 =
;;; boothreq zzz10 +
;;; calcack' zzz09 zzz13'

```

```

;;; zzz11 =
;;; calcack zzz10 +
;;; calcack' zzz11 +
;;; zzz10 zzz11

```

```

;;; zzz12 =
;;; boothreq zzz12 +
;;; calcack' zzz11 zzz13'

```

```

;;; zzz13 =
;;; calcack zzz12 +
;;; calcack' zzz13 +
;;; zzz12 zzz13

```


Appendix D. Simulation Results

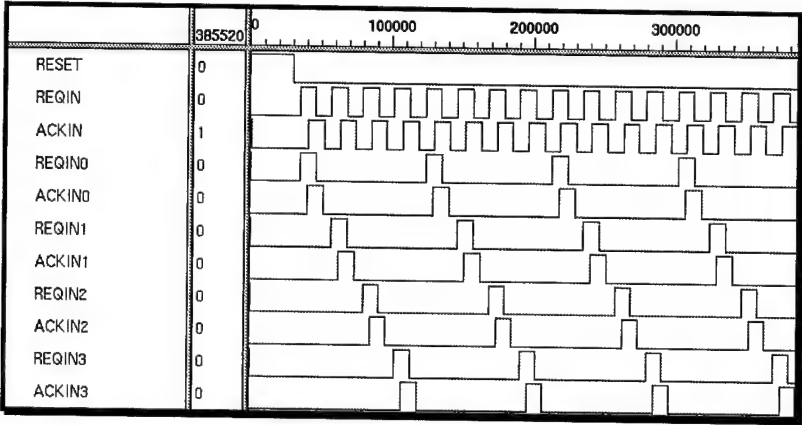


Figure D-1. VHDL Simulation of Decimator Timing

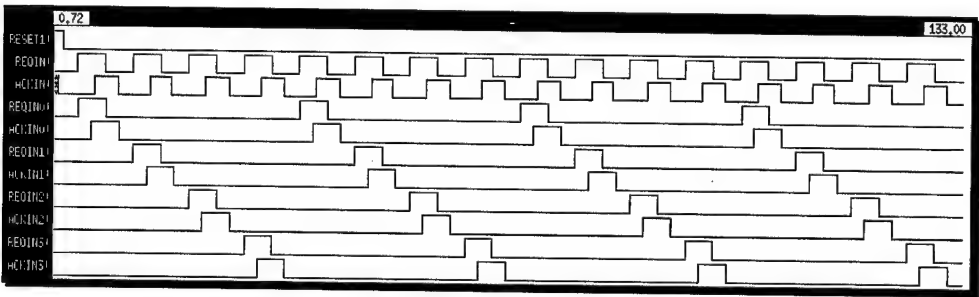


Figure D-2. IRSIM Simulation of Decimator Timing

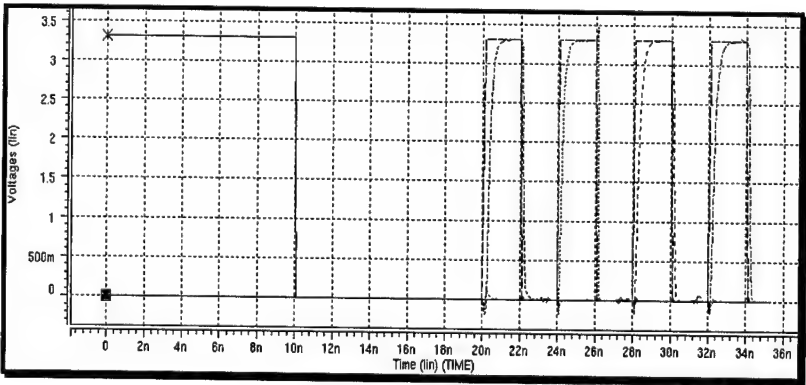


Figure D-3. HSPICE Simulation of Decimator Timing

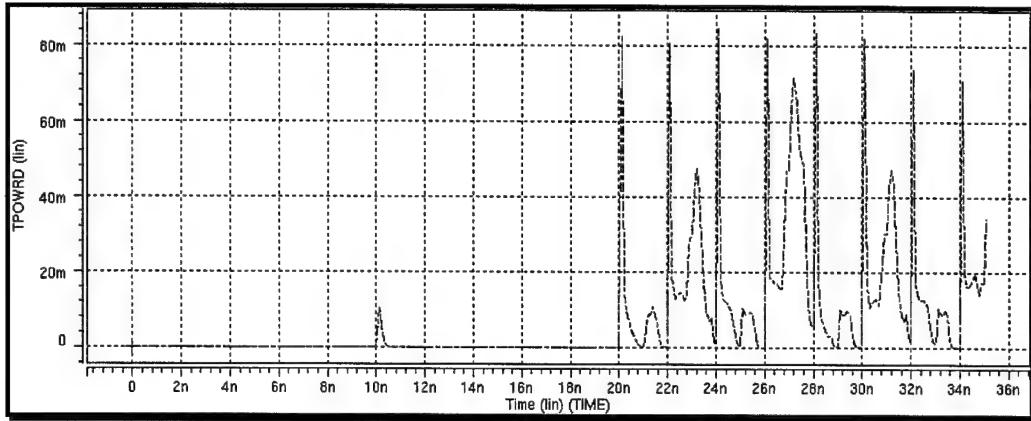


Figure D-4. HSPICE Simulation of Decimator Power

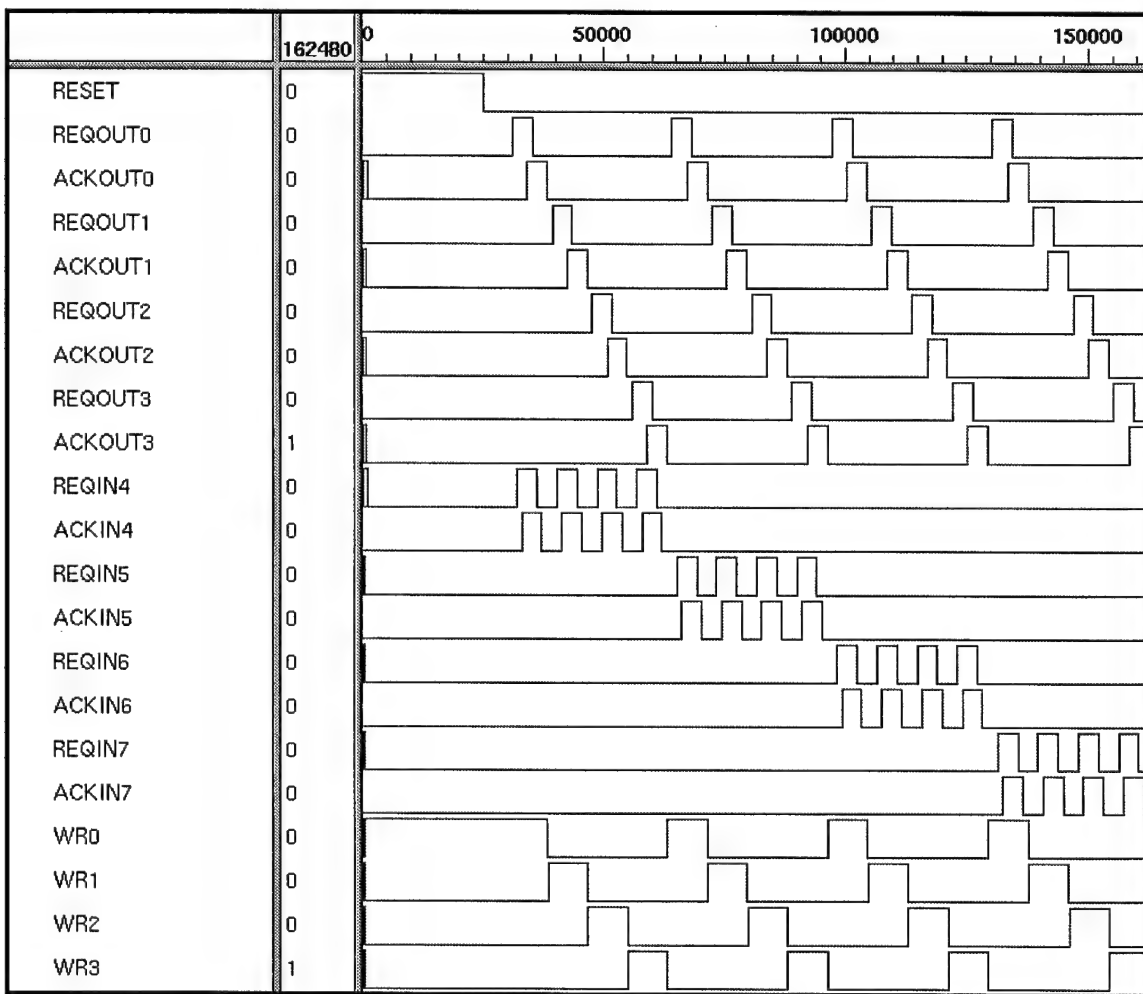


Figure D-5. VHDL Simulation of Crossbar Timing

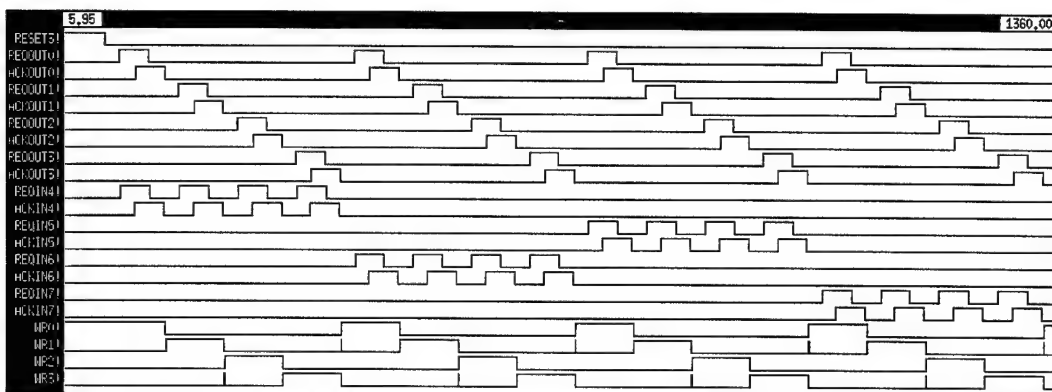


Figure D-6. IRSIM Simulation of Crossbar Timing

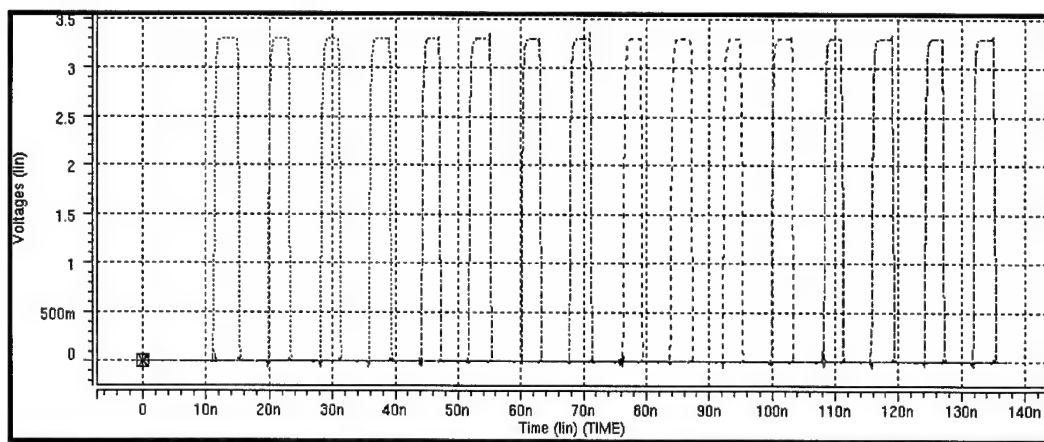


Figure D-7. HSPICE Simulation of Crossbar Timing

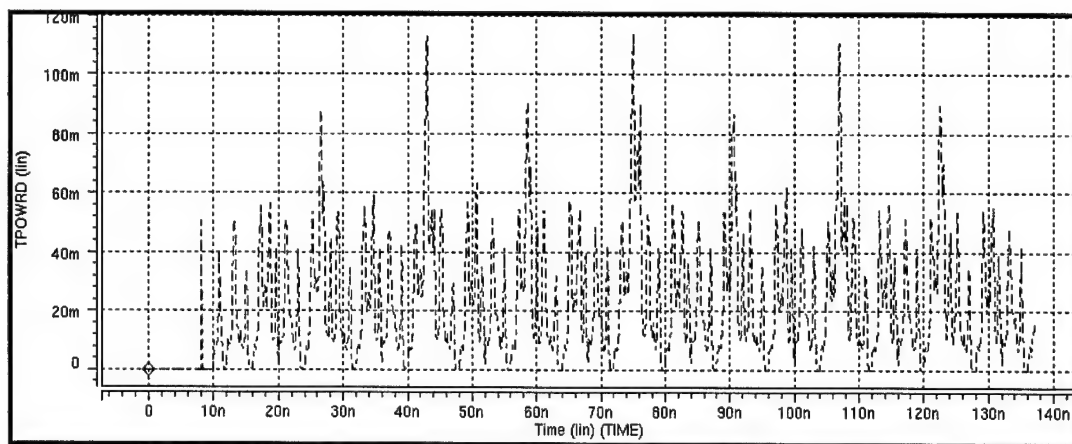


Figure D-8. HSPICE Simulation of Crossbar Power

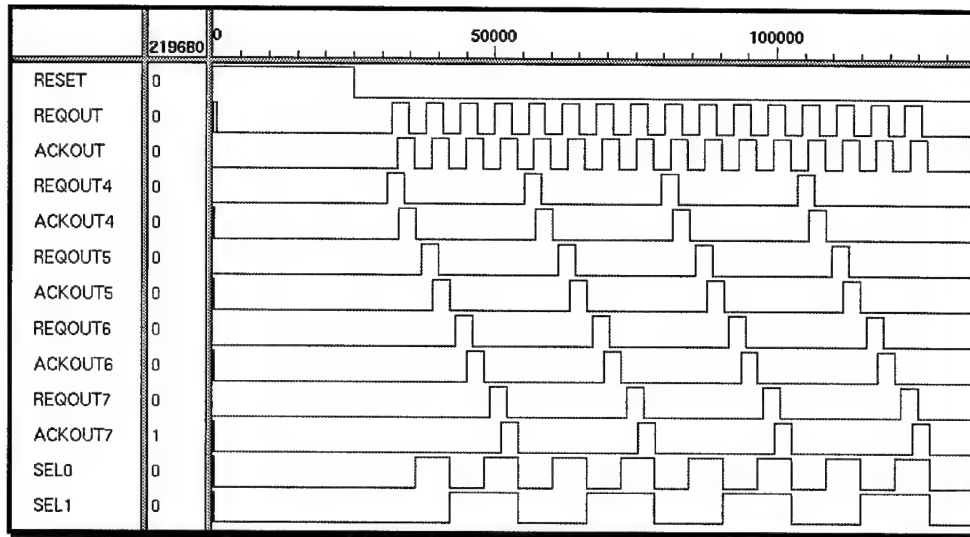


Figure D-9. VHDL Simulation of Expander Timing

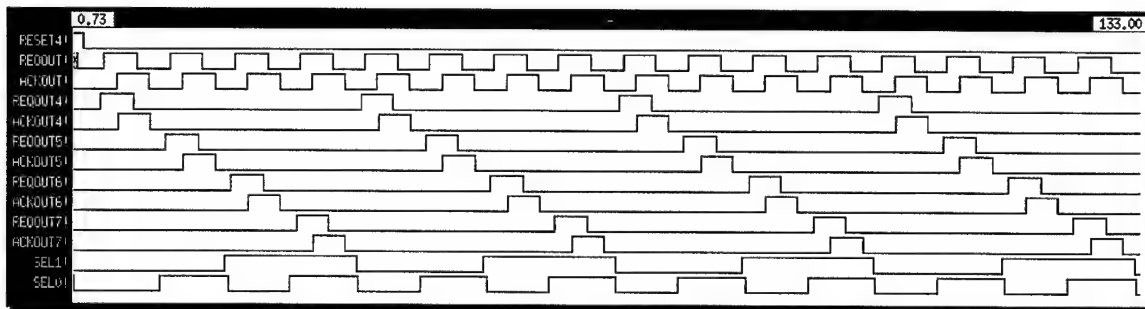


Figure D-10. IRSIM Simulation of Expander Timing

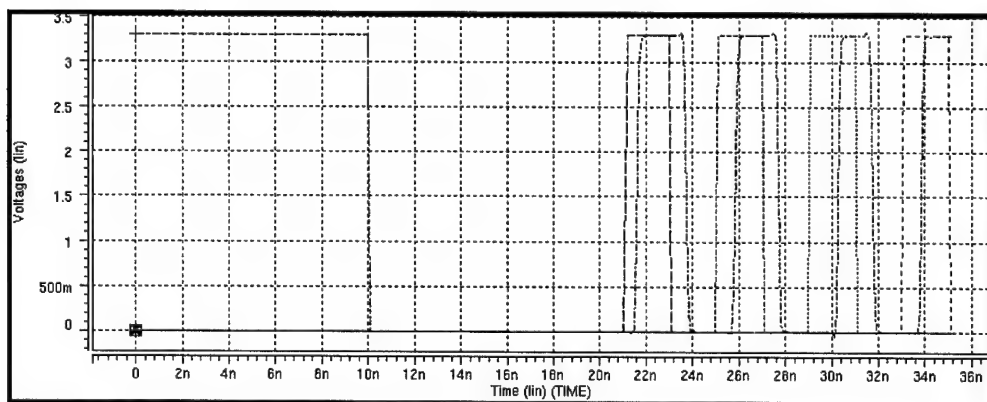


Figure D-11. HSPICE Simulation of Expander Timing

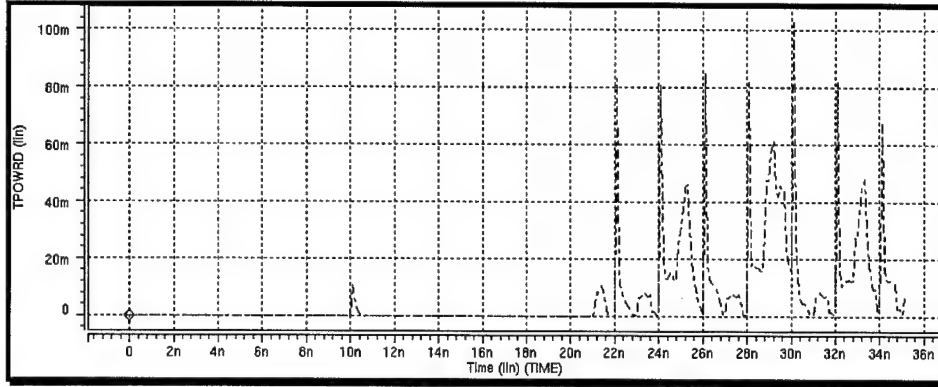


Figure D-12. HSPICE Simulation of Expander Power

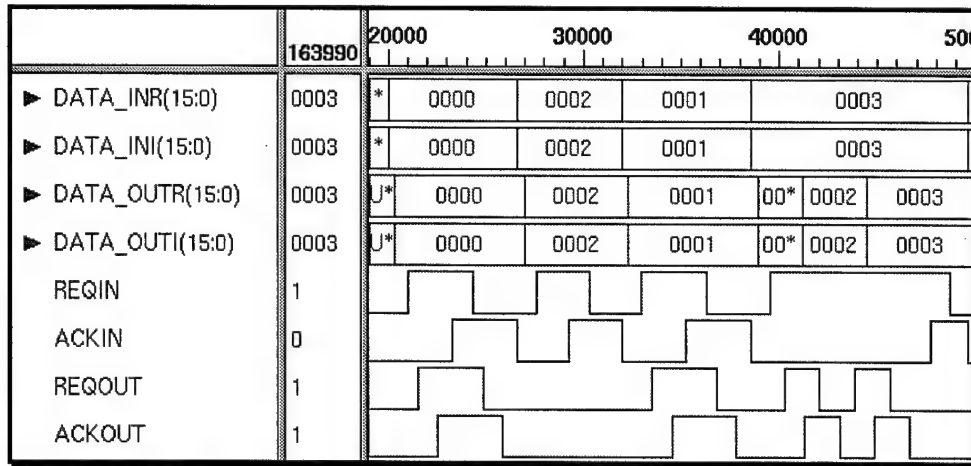


Figure D-13. VHDL Simulation of Reorder Register Timing

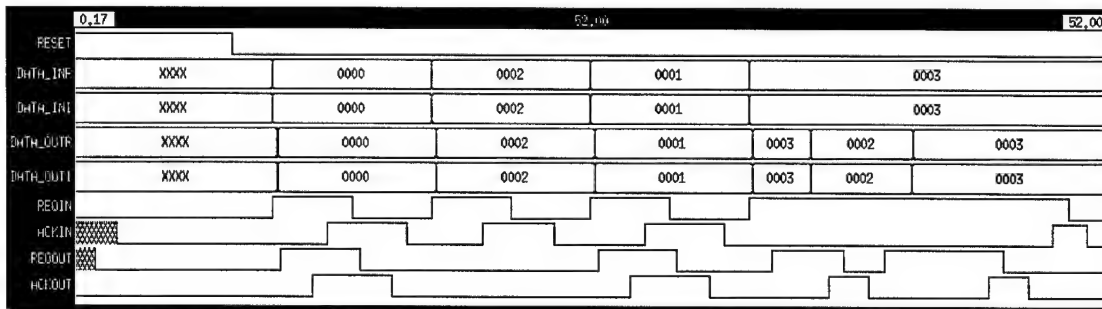


Figure D-14. IRSIM Simulation of Reorder Register Timing

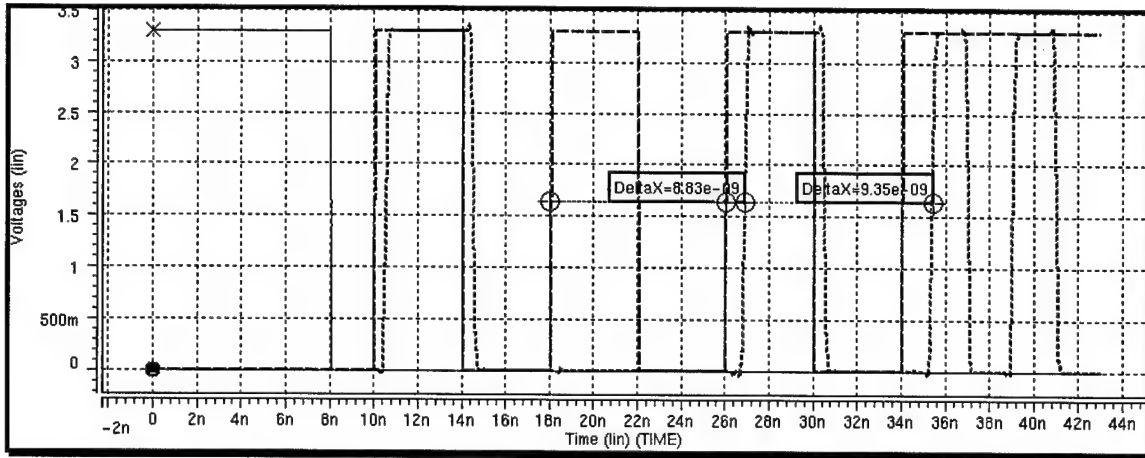


Figure D-15. HSPICE Simulation of Reorder Register Timing

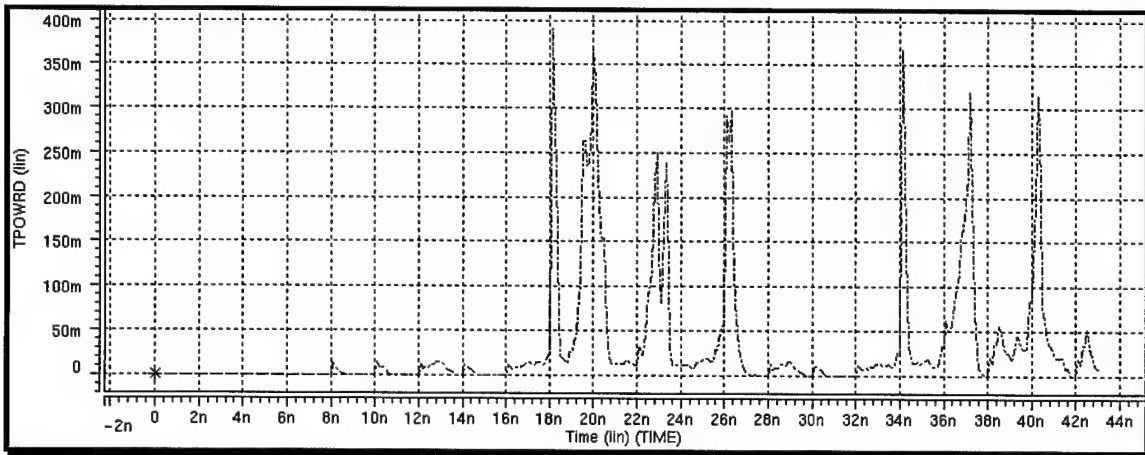


Figure D-16. HSPICE Simulation of Reorder Register Power

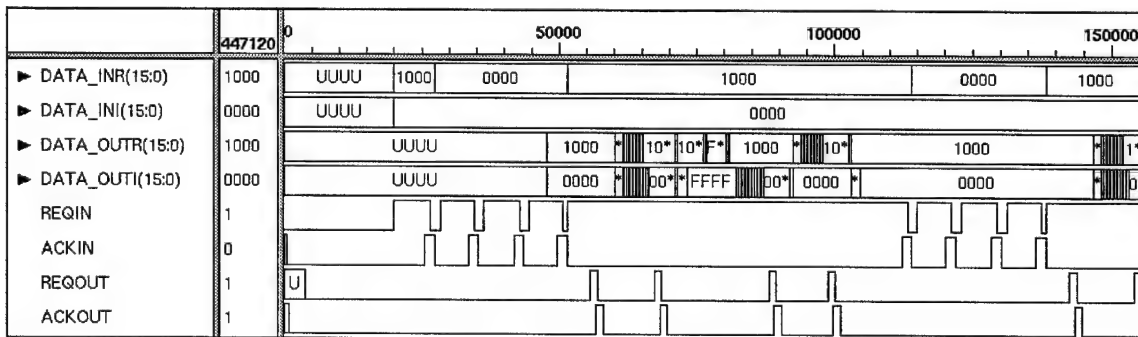


Figure D-17. VHDL Simulation of FFT-4 Timing

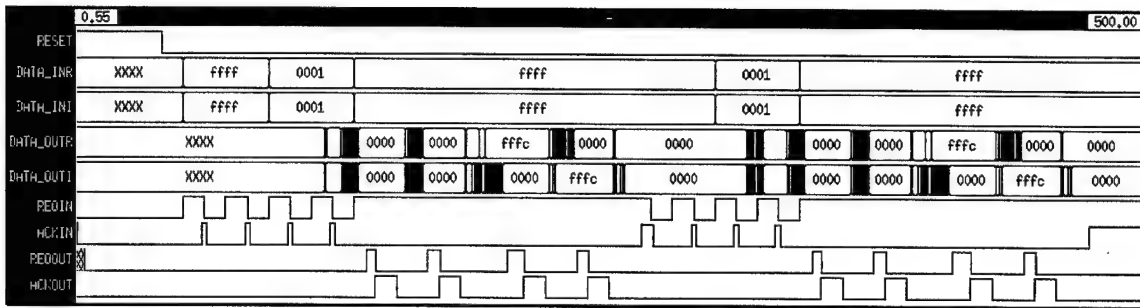


Figure D-18. IRSIM Simulation of FFT-4 Timing

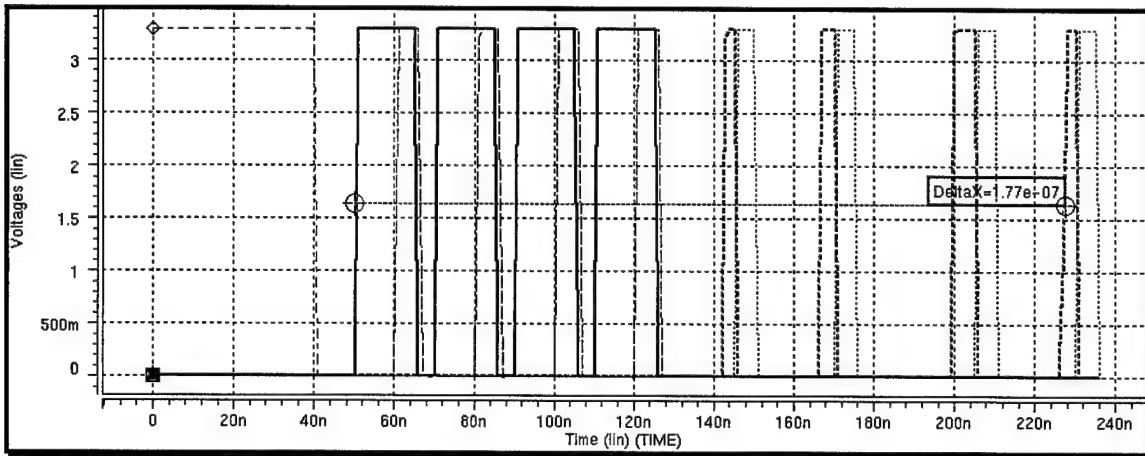


Figure D-19. HSPICE Simulation of FFT-4 Timing

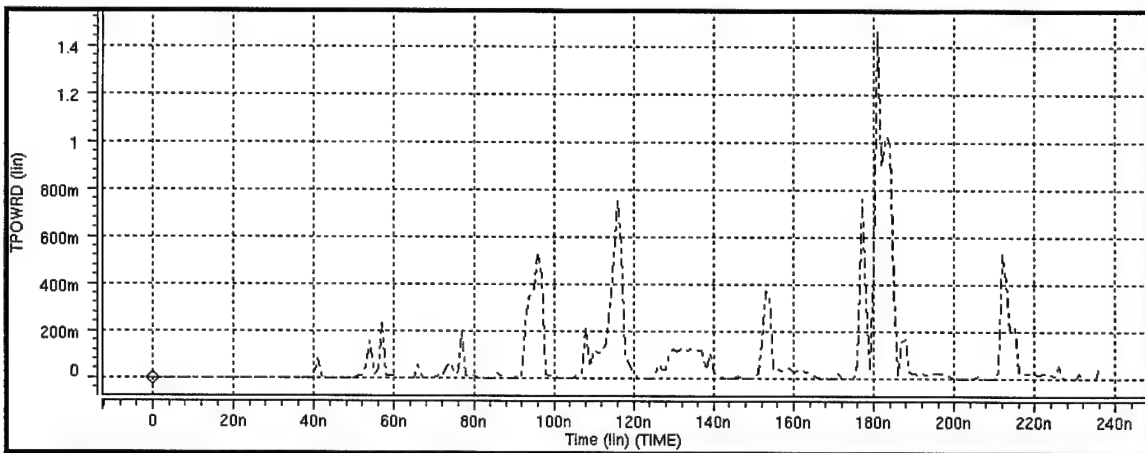


Figure D-20. HSPICE Simulation of FFT-4 Power

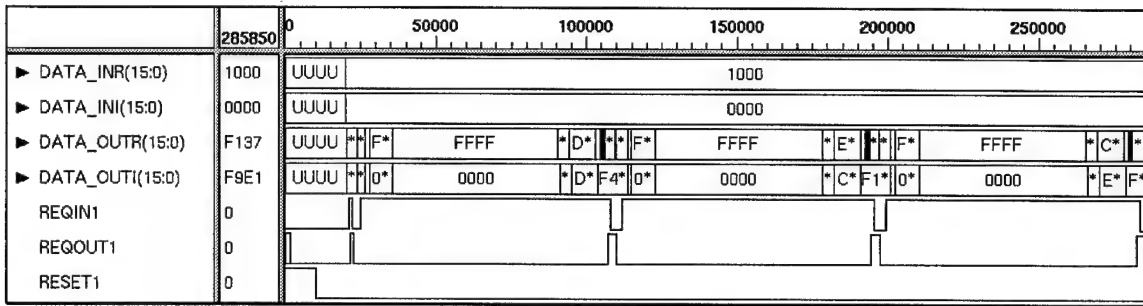


Figure D-21. VHDL Simulation of Complex Multiplier Timing

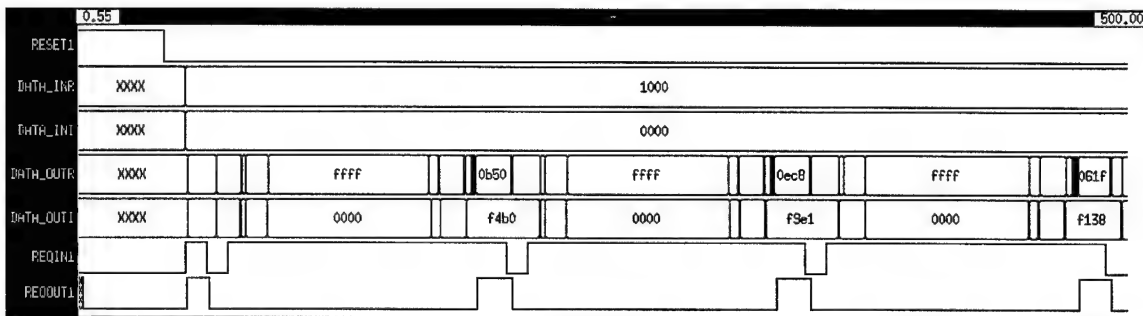


Figure D-22. IRSIM Simulation of Complex Multiplier Timing

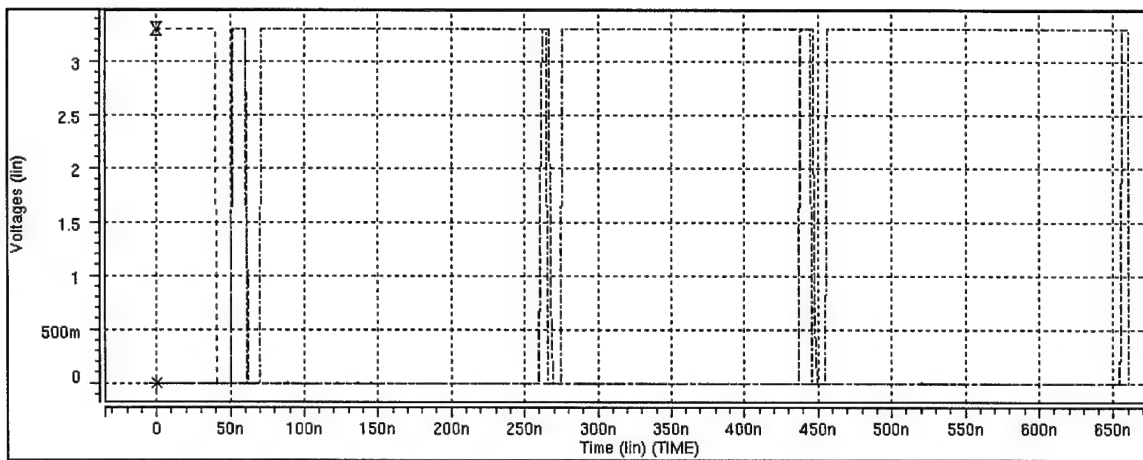


Figure D-23. HSPICE Simulation of Complex Multiplier Timing

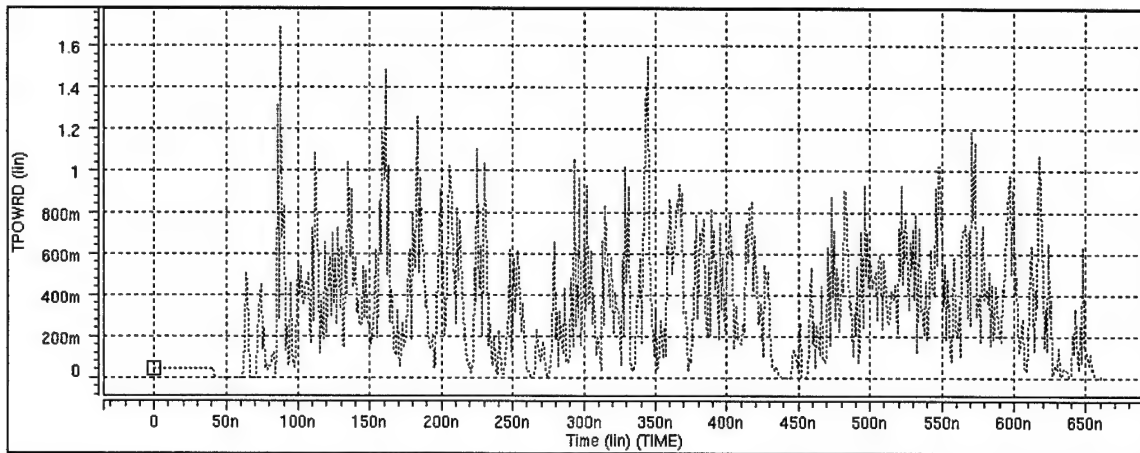


Figure D-24. HSPICE Simulation of Complex Multiplier Power

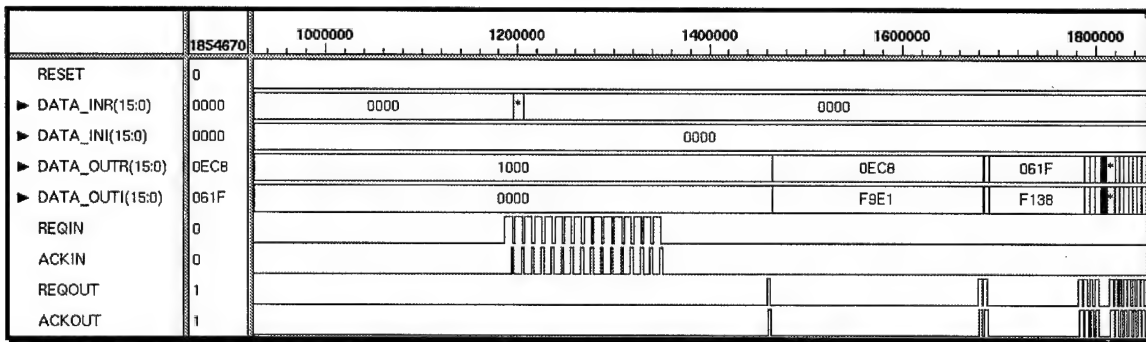


Figure D-25. VHDL Simulation of FFT-16 Timing

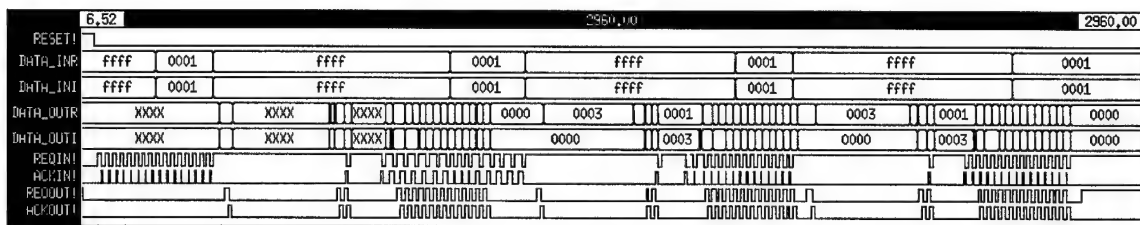


Figure D-26. IRSIM Simulation of FFT-16 Timing

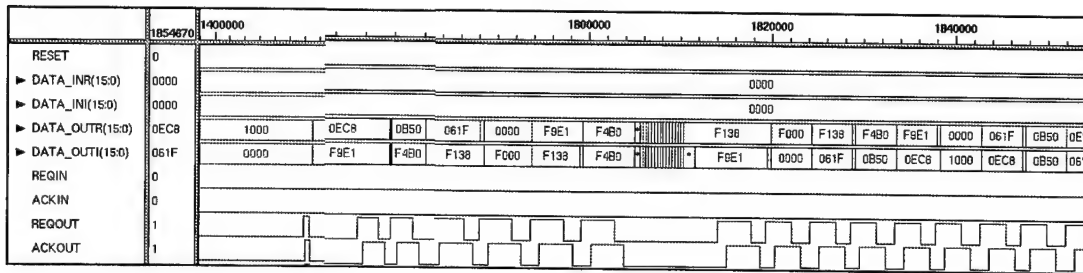


Figure D-27. FFT-16 VHDL Simulation of Impulse Response

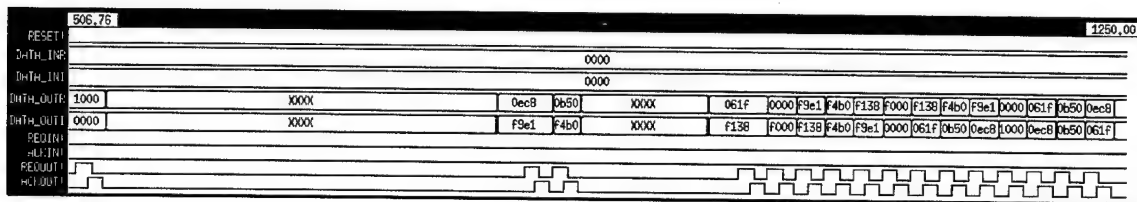


Figure D-28. FFT-16 IRSIM Simulation of Impulse Response

References

1. Suter, Bruce W. and Kenneth S. Stevens, "Low Energy Consumption, High Performance, Fast Fourier Transform." U. S. Patent Number: 08/863,239, May 1997.
2. Hunt, Bruce W. *A Single Chip Low Power Implementation of an Asynchronous FFT Algorithm for Space Applications*. MS Thesis, AFIT/GCS/ENG/97D-08. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1997.
3. Hauck, Scott. "Asynchronous Design Methodologies: An Overview," *Proceedings of the IEEE*, 83: 69-93 (January 1995).
4. Brothers, Charles P., et al. "Radiation hardening Techniques for Commercially Produced Microelectronics for Space Guidance and Control Applications," 20th *Annual American Astronautical Society Guidance and Control Conference*. 169-180. February 1997.
5. Weste, Neil H. and Kamran Eshraghian, *Principles of CMOS VLSI Design A Systems Perspective, Second Edition*. New York: Addison Wesley, 1993.
6. Hunt, B. W., K. S. Stevens, B. W. Suter and D. S. Gelosh. "A Single Chip Low Power Asynchronous Implementation of an FFT Algorithm for Space Applications," *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 216-223. 1998.
7. Morton, Shannon V., Sam S. Appleton and Michal J. Liebelt. "An Event Controlled Reconfigurable Multi-chip FFT." *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 144-153. November 1994.
8. Unger, Stephen H. *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, 1969.
9. Brzozowski, Janusz A. and Carl-Johan H. Seger. *Asynchronous Circuits*. New York: Springer-Verlag, 1995.
10. IEEE standard 1076-1993.
11. *Synopsys Design Analyzer Reference Manual*, Synopsys Incorporated, 700 East Middlefield Rd., Mountain View CA, 1998.
12. *Synopsys Graphical Environment User Guide*, Synopsys Incorporated, 700 East Middlefield Rd., Mountain View CA, 1998.

13. Shung, C. B., et al, "An Integrated CAD System for Algorithm-Specific IC Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10: 447-463 (April 1991).
14. Sutherland, Ivan E. "Micropipelines." *Communications of the ACM*, 32: 720-738. (June 1989).
15. Farnsworth, Craig, et al. "Utilising Dynamic Logic for Low Power Consumption in Asynchronous Circuits," *International Symposium on Advanced Research in Asynchronous Circuits and Systems*. 186-194. November 1994.
16. Coates, William S., et al. "Automatic Synthesis of Fast Compact Self-Timed Control Circuits," *IFIP Working Conference on Design Methodologies*. 193-208. April 1993.
17. Yun, Kenneth Y. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. Ph.D. dissertation. Stanford University, Stanford CA, 1994.
18. Raminrez, Robert W. *The FFT, Fundamentals and Concepts*. Englewood Cliffs NJ: Prentice-Hall, 1985.
19. Cooley, James W. and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, 19: 297-301 (April 1965).
20. Suter, Bruce W. and Kenneth S. Stevens. "Low Power, High Performance FFT Design," *IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics*. 99-104. June 1997.
21. Suter, Bruce W. *Multirate and Wavelet Signal Processing*. San Diego CA: Academic Press, 1997.
22. Holmes-Siedle, Andrew and Len Adams. *Handbook of Radiation Effects*. Oxford, New York: Oxford Press, 1994.
23. T. Ma and P. Dressendorfer, *Ionizing Effects in MOS Devices and Circuits*. New York: John Wiley and Sons, 1989.
24. Messenger, George C. and Milton Ash. *The Effects of Radiation on Electronic Systems*, 2nd Edition. New York: Van Nostrand Reinhold, 1992.
25. J. Osborn, R. Laco, D. Mayer, and G. Yabiku, "Total Dose Hardness of Three Commercial CMOS Microelectronics Foundries," *RADECS97 Proceedings*. 265-270. September 1997.
26. RAD 6000, Lockheed Martin Federal Systems, Manassas VA.
<http://www.lmco.com/manassas/space>

27. C40, Texas Instruments, Dallas TX.
<http://www.ti.com/sc/docs/military/liter/sgyv005c.pdf>
28. Baas, Bevan M. "And Energy-Efficient Single-Chip FFT Processor," *Symposium on VLSI Circuits*. 164-165. June 1996.
29. Sunada, Glen, et al. "COBRA: A 1.2 Million Transistor Expandable Column FFT Chip," *ICCD*. 546-550. 1994.
30. FFT Info Page, Stanford University, Stanford CA.
<http://nova.stanford.edu/~bbaas/fftinfo.html>
31. Stevens, Kenneth S. Intel Corporation, Hillsboro OR, private communication, 2 March 1999.
32. HSPICE User's Manual Version 96.1, Meta-Software, Inc., 1300 White Oaks Road, Campbell CA, February 1996.
33. Smith, Winthrop W. and Joanne Smith. *Handbook of Real-Time Fast Fourier Transforms*. New York: IEEE Press, 1995.
34. SanGregory, Sam L. *A 16-bit Asynchronous Multiplier*. CSCE-699 Project, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1996.
35. Booth, A. D. "A Signed Binary Multiplication Technique," *Quarterly Journal of Mechanics and Applied Mathematics*, 4: 236-240 (1951).
36. Mathcad7 Professional, MathSoft Inc., Denver CO, 1997.
37. The MOSIS Service, Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Ray CA 90292-6695.
<http://www.mosis.org/>

Vita

Captain David J. Barnhart was born on 16 May 1972 in Oklahoma City, Oklahoma. He graduated from John Marshall High School, Oklahoma City in 1990. Upon accepting an Air Force ROTC scholarship, he attended Oklahoma State University and graduated in 1994 with a Bachelor of Science degree in Electrical Engineering. He was subsequently commissioned a Second Lieutenant in the United States Air Force with the distinction of distinguished graduate from AFROTC Detachment 670. His reserve commission was upgraded to a regular commission in 1995.

Immediately following graduation, his first assignment was at Wright-Patterson AFB, Ohio as a Developmental Engineer in the Flight Dynamics Directorate of Wright Laboratory, now a component of the Air Force Research Laboratory. After spending two and a half years in the lab, he was selected to attend the Graduate School of Engineering, Air Force Institute of Technology (AFIT) and began his studies for a Master of Science degree in Electrical Engineering in August of 1997. He graduated from AFIT on 23 March 1999 and was assigned to the Space Electronics and Protection Branch, Space Vehicles Directorate, Air Force Research Laboratory, Kirtland AFB, New Mexico.

Permanent Address: 5612 NW 111th Street
Oklahoma City, OK 73162

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 9 Mar 99	3. REPORT TYPE AND DATES COVERED Master's Thesis Sep 98-Mar 99	
4. TITLE AND SUBTITLE AN IMPROVED ASYNCHRONOUS IMPLEMENTATION OF A FAST FOURIER TRANSFORM ARCHITECTURE FOR SPACE APPLICATIONS			5. FUNDING NUMBERS	
6. AUTHOR(S) David James Barnhart, Captain, USAF				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2950 P. Street WPAFB OH 45433-6583			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GE/ENG/99M-01	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/VSSSE Major John H. Comtois, Ph.D Bldg 914, Rm. 134 3550 Aberdeen St SE Kirtland AFB, NM 87117-5776 DSN: 246-9753			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Major Charles P. Brothers, Jr., Ph.D DSN: 785-3636 ext. 4618 Charles.Brothers@afit.af.mil				
12a. DISTRIBUTION AVAILABILITY STATEMENT Distribution Unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) A second-generation fully asynchronous Fast Fourier Transform (FFT) processor for space applications is developed in this thesis. A high-performance patented FFT architecture invented by Suter and Stevens was used as the basis for a 16-point FFT (FFT-16) processor design. A brief derivation of the architecture, the asynchronous design methodologies used and space-based integrated circuit issues are presented. The Synopsys VLSI CAD system and a radiation tolerant design library developed by the Air Force Research Laboratory were used to implement the design. A critical building block of the FFT-16, the FFT-4, was fabricated as a cost-effective method to validate the cell library and the applied asynchronous design methodologies before larger point sizes are fabricated. Results from high-fidelity simulations show that the FFT-16 design has an efficiency of 28 nJ/Unit-Transform and has a worst case throughput of 760 ns. Extrapolating these results to an FFT-1024 gives an estimated efficiency of 120 nJ/Unit-Transform and worst case throughput of 2 us. These results demonstrate that current space-based FFT processors can be replaced with a design that improves performance and efficiency by two orders of magnitude.				
14. SUBJECT TERMS VLSI, asynchronous, radiation hardened electronics, space electronics, Fast Fourier Transform, FFT, Patent#08/863,239			15. NUMBER OF PAGES 122	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	